Team Guided Parafoil System: Final Report

A Novel Method of Delivering Small Payloads from a Planetary Orbiter to the Surface

Contact email: engr-gps@uidaho.edu

Table of Contents

List of Figures

Executive Summary

In an era where rocket launches are becoming sparse, a new and innovative method is being developed to quickly and inexpensively return small scientific experiments and payloads from a planetary orbiter to the surface. A Capstone senior design team at the University of Idaho, in conjunction with engineers at NASA Ames Research Center, have been designing a passive system that utilizes a parafoil, a GPS, and winches to deliver small payloads to a pre-specified location on earth. The team successfully developed a system – complete with various wireless technologies – that can guide itself to the pre-determined location. The control unit can poll local and remote sensor packages and communicate with operators on the ground, in real time, through the Iridium Satellite Network. Novel methods were required to deploy and inflate the parafoil in low-density atmosphere. This report will summarize the project background, describe the development process of various mechanical, electrical, and software-based systems, and discuss various strategic decisions throughout the design process. This document will conclude with suggestions for future revisions of this project. Attached are appendices with technical documents and supporting reference materials.

Background

The goal of this project is to develop and improve the Guided Parafoil subsystem of the Small Payload Quick Return (SPQR) System for small sample returns from the ISS, among other locations. The SPQR system was conceived by our client Marcus Murbach at NASA Ames, and is comprised of three stages, as shown in Figure 1. The first stage uses a device called an Exo-Brake, which rapidly de-orbits the payload with a re-entry time of as little as two days. The second stage uses a Tube Deployed Re-entry Vehicle (TDRV), which stabilizes the payload and prepares it for entry into the lowest layer of the atmosphere. The focus of this project was the Stage 3 Guided Parafoil.

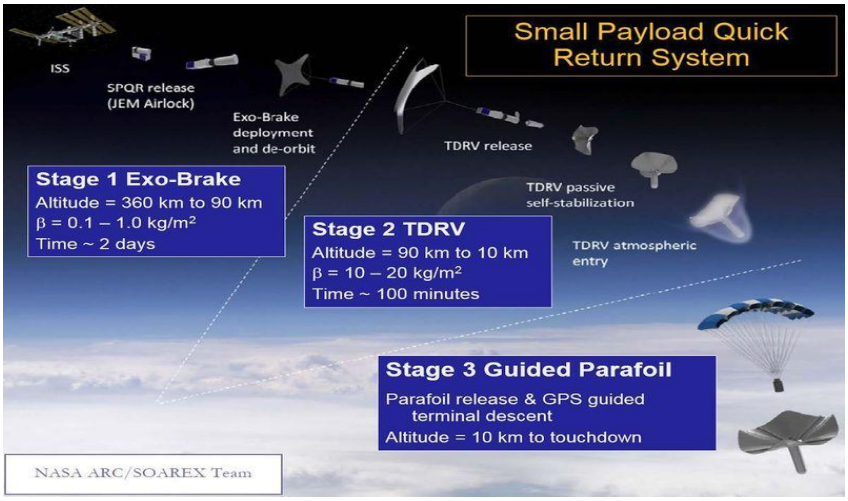


Figure 1.Small Payload Quick Return System Diagram

The Stage 3 Parafoil system has been developed by several project collaborators in the past three years. Joshua Benton, an M.S. student at San Jose State University, worked closely with Dr. Oleg Yakimenko, a professor at the Naval Postgraduate School, to integrate software flight algorithms into the Stage 3 system. His work, shown on the left side of Figure 2, included a MultiWii control board and a pair of servos for mechanical steering. His system was successfully tested on May 19, 2012. However, his system lacked wireless capabilities, so no data could be acquired in real-time.

Another project collaborator, Gabriel Pearhill, worked as an intern at NASA Ames to integrate wireless sensor technologies into the project. He used XBee Series 1 radio modules and an Iridium 9602 modem to gather telemetric data from the payload (e.g. temperature and pressure data) to send to the ground in real-time through satellite networks. His work is shown on the right side of Figure 2. Our Senior Design project was tasked with integrating the ideas from these two systems into a single system that was smaller and efficient.

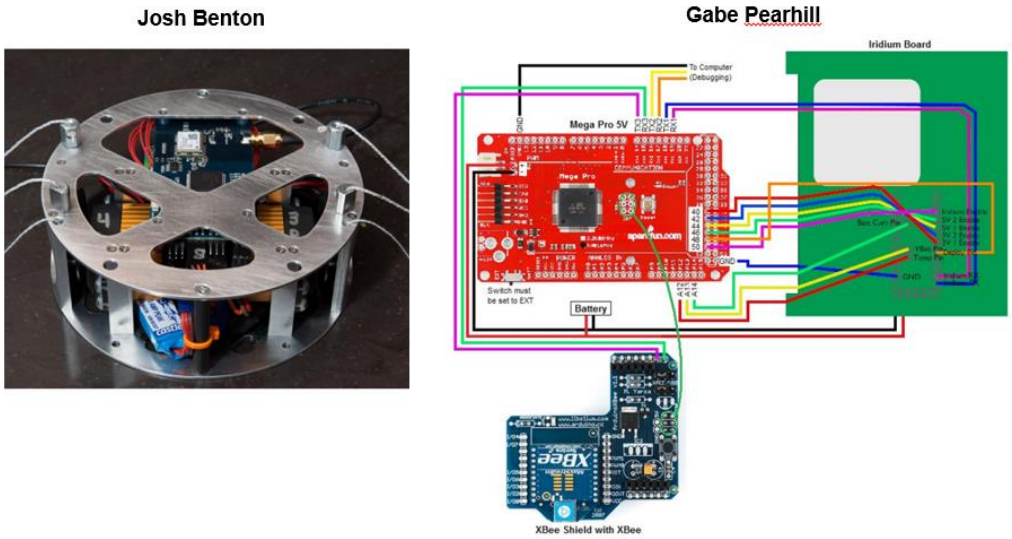


Figure 2. Previous Stage 3 Systems

Problem Definition

NASA wants to develop an inexpensive on-demand capability to return samples from the International Space Station. The Small Payload Quick Return (SPQR) system is being developed by NASA Ames Research Center to achieve this purpose. Marcus Murbach from NASA Ames tasked Team GPS to develop and improve the guided parafoil subsystems of the SPQR system. Our design objectives for upgrading the guided parafoil subsystem include:

1. Developing a parafoil deployment and inflation mechanism
2. Upgrading and shrinking the current electrical subsystems and sensors
3. Integrating XBee wireless sensor technology
4. Developing a user interface for system communication

Constraints for each design objective were identified respectively:

1. The parafoil must fit inside a 1U space and deploy in a controlled and reliable manner.
2. The microcontroller used must be as small as possible, have a reliable PWM and 12C for winch interface, and have spare I/O ports for future use. Sensors must provide a variety of telemetric data accurately and reliability under extreme climate conditions. Sensors must have high resolution, low power consumption, and have an interface with XBee network.
3. Find the best suited XBee module that minimizes size, power consumption, and cost while satisfying our range requirement of 1 meter and having 10 I/O ports.
4. The user interface must include dials of pressure, temperature, and altitude of payload. The user interface must be able to display the position of payload on a map and be accessible on all computer operating systems

Project Plan

The team responsibilities were equally allotted based on personal strengths; Stephen wrote most of the microcontroller code, lead wireless sensor research, worked on the flight control, and set the meetings agendas. Effat was responsible on improving the inflation of the parafoil in high altitude environment, solving the complex issue of the parafoil line tangling, as well as selecting appropriate servos for the guided system. Also she was the head of documentation including team minutes, poster, portfolio and keeping with the deadlines and responsible for the vacuum chamber testing. Richard and Ben collaborated on the design, structure, and implementation for the GUI, and the integration of Joshua Benton's flight algorithm. Richard created and maintained the project wiki-page. Ben acted as the team communication coordinator, and was specifically responsible for maintaining the code base and wikipage. Brian's responsibilities included anything others needed help with and other minor jobs, such as help select proper servos, and assisting in vacuum testing and drop testing. Brandon's responsibilities included selecting the sensors that were being used in the system, as well as organizing the technical presentations for the client. Jason was responsible for the design of the printed circuit board (PCB) that houses all the electrical components in the capsule. The PCB was created in DipTrace and fabricated at Advanced Circuits. It was designed to the PC/104 standards and is able to fit inside a CubeSat. Jason was also in charge of conflict management for the team. Austin designed and fabricated the CO_2-powered deployment system for ejecting the parafoil from the main capsule at high-altitudes. He was also in charge of

equipment testing, such as parafoil drop tests, deployment system tests, and the integrated system high-altitude drop test (from a weather balloon). Additionally, Austin managed the team's budget and purchasing.

Shown in Figure 3, the chronological timeline for our team throughout the year of 2014-2015.

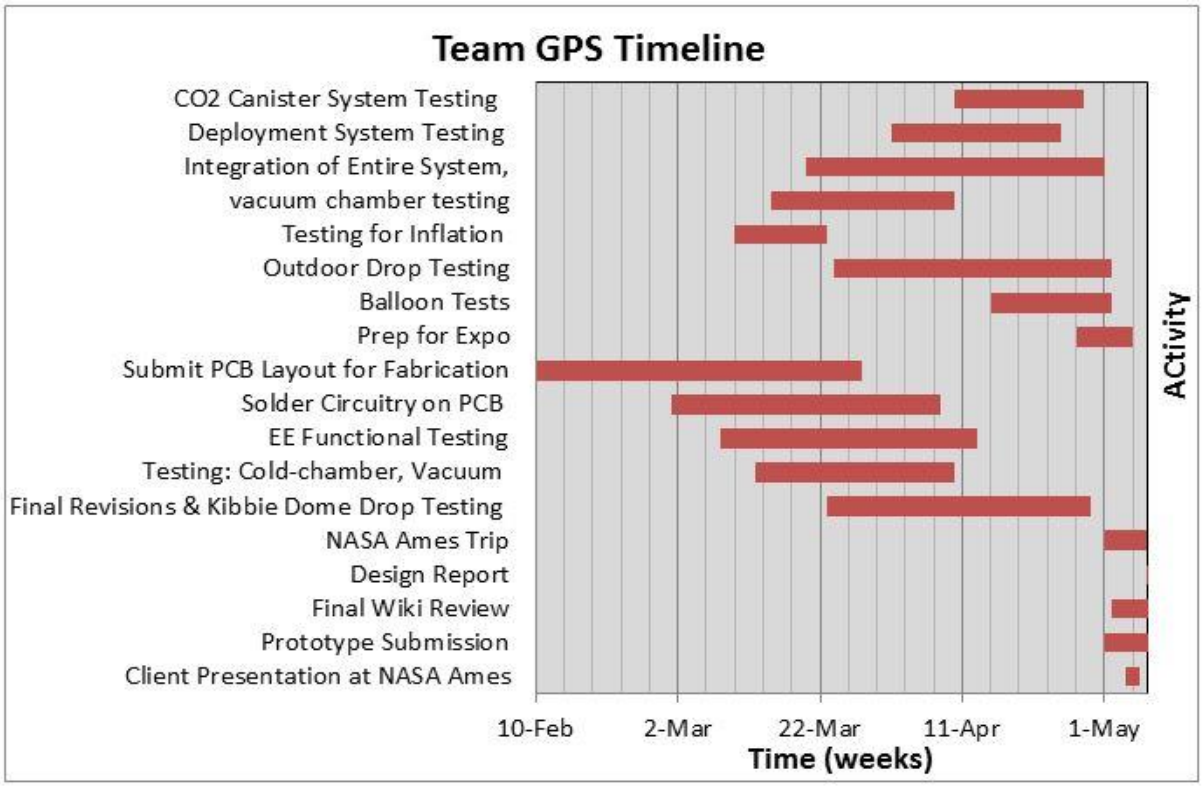


Figure 3. Gantt Chart for Team GPS

Concepts Considered

The electronic system was originally designed around a five-volt (5v) Arduino Mega Pro microcontroller, however upon further research the team moved to a 3.3v Teensy 3.1 microcontroller, as it is much smaller and more responsive. Switching to a smaller device allowed the entire system to be shrunk and housed on a PC/104 printed circuit board, rather than a breadboard. Figure 4 shows a rough comparison in size between the Teensy 3.1 and Arduino Mega. Before deciding to integrate the entire electronic subsystem on one board, the devices would have been strategically placed inside of the control structure to optimize size and durability. Switching from a 5v to a 3.3v system required the team to switch to a different microSD card breakout board as well. The team also discussed various technologies to implement an array of wireless sensors. The team began using two XBee Series 1 radios to

wirelessly transmit temperature data to the system, but then decided to move to XBee Series 2, which uses a more robust wireless network. There are three models of the Series 2 radios to choose from: regular, Pro, and Pro Programmable. The Pro is larger than the regular, has a longer range, and consumes more power. The Pro Programmable implemented an auxiliary processor onto a Pro device to allow local data processing and more extensive communication (such as SPI and I^2C). We ultimately chose to use four XBee Series 2 regular modules, as we did not need the extra functionality from more advanced models. This allowed the sensor packages to remain small while reliably transmitting pertinent data. Each of the sensor packages regularly polled an analog temperature sensor, and an analog pressure sensor. We initially chose to use a TMP36 3.3v temperature sensor, however we later moved to a LM19 3.3v temperature sensor as it has more range of output, corresponding to a wider range of measurable temperatures.

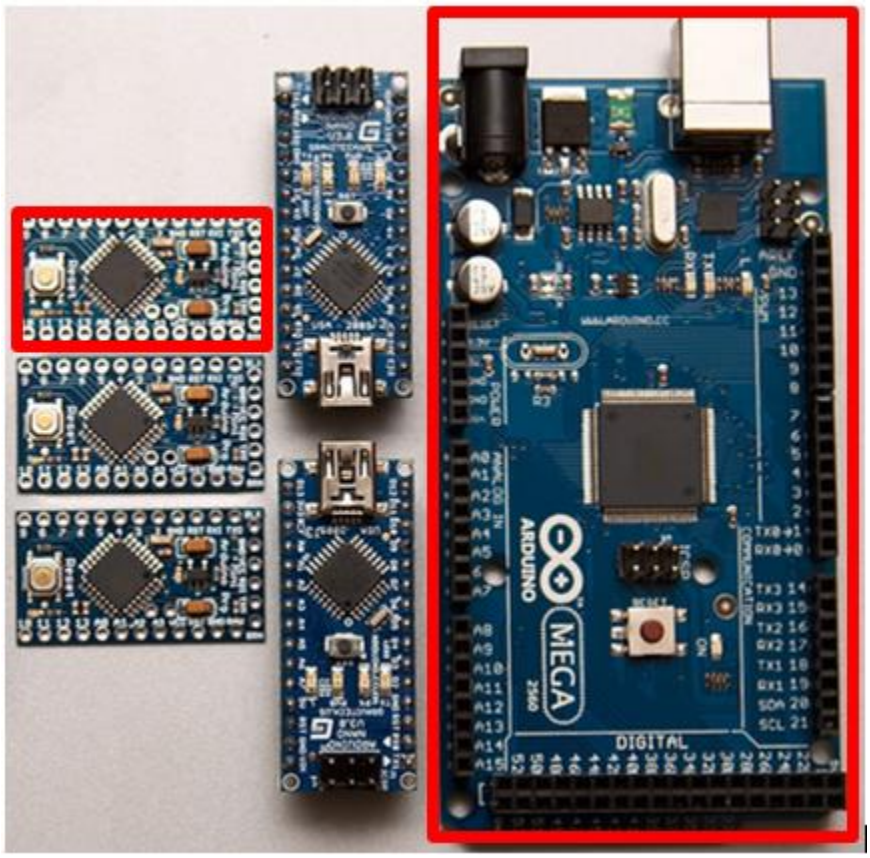


Figure 4. Microcontroller Size Comparison

We had to ensure that we were meeting the following design requirements when considering our implementation:

- Email parser - Prepares information by tokenizing data fields received via email attachments.
- Data table - Structure which stores parsed data for easy retrieval.
- GUI - Flat, clean, non-convoluted design.

To address the requirement of an email parser, we investigated our options for retrieving data. Our data source was the Iridium modem broadcasting from the parafoil. Previous implementations used Gmail attachments to deliver strings that could be parsed for desired data; this inferred a need for some sort of backend parser. Emails with a specific data format are sent from Iridium, to a specified Gmail inbox, either upon request or at regular intervals. We considered using the official Gmail API retrieve and manage messages, but found that the interface was convoluted and bloated for our purposes.

The design goal of making a clean and usable interface optimized for mobile devices led us to initially develop in Android Studio. We built a graphical prototype using Android Studio that met our requirements of displaying data that would update on command. We found that Android Studio was not ideal for implementing other features, such as maps. Android Studio is excellent for prototyping, but ran slowly and was difficult to navigate. The team developing the Kickshot game had similar issues with Android Studio, and rewrote their code using a tool called PandaJS, which incorporated JavaScript, HTML5, and css. Writing an app for Android devices only seemed limiting; whereas moving to JavaScript, HTML5, and CSS would give us the added bonus of being platform independent. Learning from their experience, we moved to HTML5.

We started with CSS, pure HTML5, and AngularJS to begin prototyping the GUI. One concern with developing a web application is the large number of different sized devices that will have access to the app. It is important to develop the app in a way that each of these devices can use the app as intended. This responsive approach to development was achieved through the use of CSS and HTML5. AngularJS was going to be used to handle the communication between the view and the controller. But after some research we found that AngularJS was more than we needed for our design requirements. All of our research led us to using a Python module named Flask. Flask acts as the server and also handles the communication to and from the client.

Then, regarding the Flight Algorithm Implementation; we asserted that Joshua Benton's flight algorithm was correct, based on his paper and conclusions from his data. Our first intention was to completely rewrite his code, as it did not fit our coding standards and was highly coupled with his own Snowflake Parafoil. We decided it would not be good to disrupt the integrity of his calculations, and set out to simply make his code more readable and maintainable. His code base was also missing a referenced "pwm.h" file, which controlled the frequency of the algorithm execution, and contained missing data structures. We considered implementing a fault-error model which would return specific error codes to aid in configuration and debugging.

Concept Selection

The selection of servos was based on a certain criteria of the following: weight, dimensions, operating voltage range, maximum rotation angle, and the stall torque. We narrowed down our choices into three servos; *Vigor VSD-22YMB, Eurgle 16D_22YMB,* and *GWS S125 6TD*. Our final choice was the *GWS S125 6TD servo,* as it was the only one available to be purchased; with a weight of 50 g, size of 40.5 x 20 x 42 mm^3, 4.8V - 6V operating voltage, and 2160^o of rotation.

The fiberglass polyester rods were chosen based on the type of commercial rods used for the automobile collapsible sun shades, this particular material facilities desired properties such as: not electrically conductive, offer a high strength-to-weight ratio, flexible, extremely durable, resistant to corrosion, low thermal expansion, and non-magnetic. The Teflon coated fishing lines were chosen based on similar applications. They have high strength-to-diameter ratio with Dupont's Teflon surface protector coating that provides enhanced abrasion resistance, ultra smooth and Hydrophobic Vibration reduction total. The deployment system represents the use of easily attainable CO2 Cartridges. The choice of this system was based on previous uses that has been proven in fields such as similar systems on high-altitude balloons, amateur rocketry, and UAV's

One of the main design goals was to shrink the current control structure of the guided parafoil system. The current control structure was a one story cylinder 5.45 inches (13.843cm) in diameter and 1.8 inches (4.572 cm) in height. However, shrinking this control structure proved difficult due to the size of the PCB board used to include the added electronic features. The length and width of the PCB board 9.6 cm x 9cm (3.54 in x 3.78 in) did not allow any room to shorten the diameter of the control structure. In addition, because of the PCB boards dimensions it would be impossible to have the support struts for the servos guiding the parafoil to be supported at both the floor and roof of the control structure as the supports would intersect the plane of the PCB board. Thus, rather than compromising structural integrity by only attaching the support struts to the floor of the control structure, I chose to build two stories within the control structure: one story for the servos and support struts and one story for the PCB board.

Many of the electronic components were pre-selected (before Senior Design began) so that they would work with other systems that NASA is developing. The code was written to be modular and easily ported to other C/C++ based microcontrollers (rather than based on Arduino-specific functionality). This allows the driver files to be moved between multiple programs or systems that need the specific hardware. The XBee code was written to communicate directly with the Coordinator API – without the use of specific XBee libraries – so that it can be ported to a non-Arduino-based microcontroller. Moving from an Arduino Mega to a Teensy 3.1, along with designing a printed circuit board, greatly simplified the design process and allowed a smaller, faster, more reliable final product.

The Graphical User Interface design selection was after a few iterations through a prototyping process, first with Android Studio then with AngularJS, we finally landed on using the Python module, Flask, combined with HTML5, CSS, and JavaScript. Using these tools we were able to create a web application that displayed dynamic gauges, a real-time flight map, and other useful diagnostic data.

There were two approaches for integration and implementation of Joshua Benton's code: A complete rewrite of the algorithm code, or implementing a wrapper function to interface between the algorithm and the rest of the code base. We chose to write a wrapper function, as this allowed us to encapsulate global variables, and control scope. We were able to write proper "getters" and "setters" while maintaining the integrity of Benton's original, tested code.

System Architecture

The final control structure built is a two story cylinder 5.45 in (13.843cm) in diameter and 2.3622 in (6 cm) in height. The second story houses the PCB board and the top and middle plate are held together by four columns drilled and tapped at each end. The first story houses the batteries for the PCB board and servos and the servos themselves. The bottom and middle plate are held together by four struts drilled and tapped at each end which provide support and attachment for the servos in addition to two columns drilled and tapped at each end. To provide additional support for the servos pulley housings were built and are also located on the first story of the control structure. An important feature put into the design of the control structure is that the middle and top plates are identical, thus making manufacturing easy as the two plates can be built at the same time. Ultimately, the control structure did not satisfy the design requirement of shrinking the control structure as the diameter stayed constant and the control structure had to increase in height by 1.5 cm due to the PCB size as discussed in the concept selection. However, under future work for this unit it is discussed on how to shorten the height and thus meet the design requirement. The justification for the continued development of the control structure after we knew we had to increase the height slightly instead of shrinking it, is that we needed a control structure to launch, the added height is still a good trade off compared to the added electronic features, and the control structure still fits within the PCTCU unit developed by NASA. Drawings for the control plates, struts, columns, and pulley housing are provided in the appendices.

The electrical subsystem is centered on a student-designed printed circuit board (PCB) and a Teensy 3.1 microcontroller. The PCB, pictured in Figure 5, replaces a breadboard, enabling the system to be smaller, more durable, and easier to debug. The PCB was designed to PC/104 standards, allowing the system to be applied to CubeSats in future revisions.

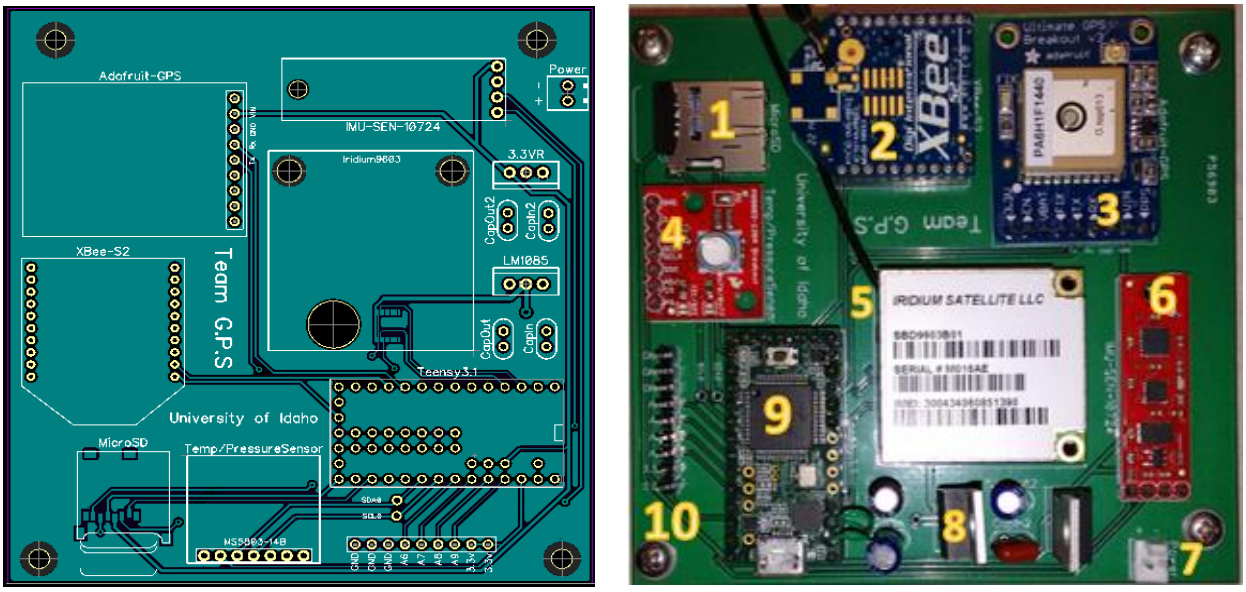


Figure 5. PCB Layout

The electrical subsystem consists of a microSD card to store data locally (1), a Series 2 XBee radio to send and receive data from wireless sensor packages (2), an Adafruit GPS receiver (3), a Sparkfun digital pressure and temperature sensor (4), an Iridium 9603 data modem with antenna (5), a Sparkfun Inertial Measurement Unit (6), a 7.4 volt (V) polarized battery connection (7), power electronics (8), a Teensy 3.1 microcontroller (9), and breakout pins to connect external analog sensors (10). The power electronics convert the 7.4v input into either 3.3v or 5v for the various components, and the Teensy 3.1 communicates with each individual package, and stores the data it receives. The Iridium modem, GPS receiver, and XBee radio communicate with the Teensy 3.1 with a 3-line serial connection (Rx, Tx, GND), while the digital temperature and pressure sensor, along with the inertial measurement unit, communicate with the Inter-Integrated Circuit (I^2C) protocol. The Teensy writes data to the microSD card using the faster Serial Peripheral Interface (SPI) bus. All of the components utilize 3.3v logic, however the Teensy 3.1 is also capable of 5v logic if necessary.

The code is modularized, with separate driver files for many of the individual components. This allows for easy integration of devices across a wide array of systems while simplifying the debugging process. To integrate a component onto a new system, or replace components on the current system, the developer only needs to copy the driver file into the code base for the new system and call the student-written wrapper functions to interact with the device. The code was developed in C++ in the Arduino integrated development environment (IDE), however most of it was written to interact with the low-level hardware, rather than

requiring Arduino-specific libraries, so it can easily be ported to any C-based embedded system (such as non-Arduino microcontrollers). The control and data-logger functions utilize a non-blocking delay, allowing the developer to poll the components at an arbitrary frequency without interrupting other sections of the code.

The electronic subsystem also includes two winches which can be controlled to steer the system to a pre-specified latitude and longitude, based upon its current GPS coordinates. The winches are controlled with pulse-width modulation (PWM), and are enabled only when the system is falling. A system is in place to avoid false positives, thus preventing the system from attempting to navigate while it is rising.

The parafoil deployment system is an aluminum canister that houses a 12g CO_2 cartridge, a spring, a nylon spring holder, a sharpened set screw, two stainless steel plugs (one acting as a firing pan and the other as a plunger), and two O-rings (used for sealing between the plugs and the inside of the canister wall). The sharpened set screw is inserted into one end of the canister and the spring, spring holder, and CO_2 cartridge are then also inserted. The spring holder aligns the spring with the CO_2 cartridge with the spring holding the cartridge away from the set screw to keep it from puncturing prematurely. Next, the two stainless steel plugs are inserted. A Pyrodex charge (0.2mL ~ 1.5g) is placed between the two plugs along with a model rocket igniter (Estes Starters were used in this system). The entire system is actuated by running a current through the model rocket igniter (a 7.6v battery was used as a power source in this system). This ignites the Pyrodex, causing the upper stainless steel plug to push the CO_2 cartridge into the sharpened set screw. The released gas is directed through ports in the top of the canister, pushing the parafoil out of the capsule. Engineering drawings of the deployment system and its components are provided in the appendices.

The Graphical User Interface (GUI) is a dynamic web application that scales for both desktop and mobile viewing. Its design follows a Model-View-Controller (MVC) pattern, and is implemented in Python, Flask, and Javascript.

- Model
    - GPS data received from Iridium is stored in a json struct. ‘Latitude’, ‘Longitude’, ‘Altitude’, ‘Time’, and ‘Satellites’ fields contain parsed data which update the corresponding text boxes and gauges in the view.
    - Gmail contains email messages, which store information received from Iridium in attachments, as custom formatted GPS strings.
- View
    - The main view is served by Flask. It is dynamic so scales with window size and accommodates desktop and mobile browsers.
    - The gauges and corresponding fields update synchronously.
    - Google maps is used to display location based on latitude and longitude.
    - Layout is a combination of html and css.

- Controller
    - *main.js* contains a function *get_data()*, which is the main controller. This function periodically points to an undisplayed URL, which runs a parsing routine.
    - Gmail is interfaced via a python imap library, and parsing is performed to extract the desired data. New emails containing updated information are found, opened, and data is stored for the *get_data()* routine.

The source code for Josh Benton's flight algorithm is tightly coupled with his own Snowflake parafoil design. We reformatted but did not change his code in order to retain its integrity. A wrapper function, *flight_control()*, was developed in order to interface with the existing code, but not disrupt proper calculation of flight control values. Additional string parsing, conversion of latitude and longitude values, and additional structs were written to make integration with the rest of the system possible.

Testing

This project required multiple on-the-ground system tests. These tests included XBee communication tests, GUI testing, Iridium data sending/receiving tests, and deployment system tests. These tests generally consisted of the system being put through some normal operations to see if it could handle it. For example, the deployment system was tested using a different parafoil and a different capsule than the ones originally planned for it; however, the tests were to see if the system would even come close to working. It was shown to work quite well in the tests; however, due to time constraints, it was never tested with the fully integrated system.

Because of the nature of this project, not only were normal on-the-ground system tests necessary, but low-altitude and high-altitude drop tests were also needed. The tricky part about performing drop tests, whether high or low altitude, is that you have to have a suitable venue. For the low-altitude parafoil drop tests, the University of Idaho Kibbie Dome was selected. Dropping the parafoil from the second level provided for roughly 100 ft of drop. This was more than sufficient for initial parafoil direction and inflation tests. For the final integrated high-altitude drop, a 1600g weather balloon was used. This weather balloon was easily capable of taking the payload to altitudes in excess of 70,000 ft, however a target altitude of 50,000 ft was used because it had lower the risks and decreased the drift distance of the parafoil. Sadly, this high-altitude test was cut short when the parafoil was released early. An equipment malfunction onboard the balloon train caused the parafoil system to be cut down at roughly 4,000 ft. This only allowed the system to collect a couple of minutes of flight data; however it, did result in some useful information collected.

Budget Summary

The senior design team was given a budget of $8000. At the beginning of the first semester, the team wrote a cost estimate sheet outlining where the money would be going. This initial cost estimate sheet is included in the Appendices. The three major expense areas were: (1) materials, e.g. aluminum rod, PCB, sensors, (2) testing, e.g. balloons, helium, vehicles, and (3) the trip to Ames for the team's final presentation. Thankfully, the team was able to finish under budget in almost all of these areas, i.e. materials and the trip to Ames.

Future Work

It's recommended to use the 22.7 kg (50 lb), 0.36 mm (0.014 in) diameter P-Line TCB 8 Teflon Coated 8-Carrier Braided Line for the parafoil lines as they have the best properties to eliminate tangling. Also, it's best to use the Vigor VSD-22YMB for the servos. The control structure unit's height will be decreased by ~ 1 cm. Future work on the control structure would be to lower the second story of it. This would be easily possible if surface mounting voltage regulators were used instead on the PCB board. The effort would not be more than a day's work, as the only thing that needs changing is the height of the support columns connecting the middle and top plate. Additionally, the columns securing the servos to the control structure could be modified to be U-shaped "saddles" that the servo would sit into. These "saddles" would only be connected to the bottom board which would allow room above the servos (since the columns wouldn't stick out above them anymore) for the PCB to slide in, essentially eliminating the need for a second story.

A thermal control system could be added to regulate the capsule internal temperature while the system is in flight, and the flight algorithm could be further tested and improved. The code is written to allow for commands to be sent from the graphical user interface (GUI) to the control structure while the system is in flight. Enabling this functionality could allow future students to develop a useful set of commands to implement while in flight. Concept features could include cutting the payload down, putting the system to sleep to conserve power, or managing a future on-board thermal control system (increase or decrease internal temperature). The microcontroller could also send AT commands to individual XBee sensor modules to enable/disable additional input or output (I/O) pins, change the sleep mode, or modify the sensor polling frequency. Finally, the printed circuit board could be redesigned with more surface mount components (rather than through-hole), and with the electronics - such as processors and digital devices - mounted directly on the board, rather than on breakout boards that are soldered to the board. Additionally, the SPQR system that NASA is designing is run off of 5v batteries, so this system should be changed entirely over to 5v.

The GUI is functional and definitely improves the quality of diagnostic data viewing, however there are quite a few changes that could be made.

- Adding derived calculations like heading, velocity, change in altitude, etc.
- Add functionality that allows a user to view past launch data in real-time.
    - This data would be downloaded and stored in a database after the initial viewing.
    - The user could use a slider to view each packet at their discretion.
- Implement an admin user login get rid of the username and passwords that are stored in plaintext in the code.
    - The admin user will be able to input the username and password from the app itself.

The flight algorithm is now in a state that it can be easily modified and repurposed. Ideally, future work should include:

- Error code implementation and logging:
    - This could be implemented by writing all functions/methods to return an int, and use those codes to log specific types of errors.
- A configuration GUI, that would set crucial starting parameters such as target latitude, longitude, and altitude.

Appendix A. Microcontroller Code

```
/* ************************************************************************
 * File Name:       XBeeDigital.c
 * Author:          Stephen Wayne
 * Created:         December 29, 2014
 * Revised:         September 4, 2014

 * Project Description: This is the driver file for an API Coordinator
 * XBee Series 2 module.
 ************************************************************************ */

//#include <plib.h>
#include <WProgram.h>

#include "XbeeIO.h"

void init_XBee(){ // initialize XBee
  XBee_Serial.begin(XBEE_DATA_RATE);
}

byte digi_check(word digi_mask){ // Check if digital pins enabled
 byte isDigital = 0; // reinitialize counter
 if(digi_mask & DIO0){
   Serial.println("DIO0");
   isDigital++;
 }
 if(digi_mask & DIO1){
```

```
  Serial.println("DIO1");
  isDigital++;
}
 if(digi_mask & DIO2){
  Serial.println("DIO2");
  isDigital++;
}
if(digi_mask & DIO3){
  Serial.println("DIO3");
  isDigital++;
}
if(digi_mask & DIO4){
  Serial.println("DIO4");
  isDigital++;
}
if(digi_mask & DIO5){
  Serial.println("DIO5");
  isDigital++;
}
if(digi_mask & DIO6){
  Serial.println("DIO6");
  isDigital++;
}
if(digi_mask & DIO7){
  Serial.println("DIO7");
  isDigital++;
}
if(digi_mask & DIO10){
```

```
    Serial.println("DIO10");
    isDigital++;
  }
  if(digi_mask & DIO11){
    Serial.println("DIO11");
    isDigital++;
  }
  if(digi_mask & DIO12){
    Serial.println("DIO12");
    isDigital++;
  }
  if(!digi_mask){
    Serial.println("No Digital IO");
    isDigital = 0;
  }
  return isDigital;
}

// For Debugging purposes only. Serial output for which
// ADC pins are enabled on each XBee
void ana_check(byte ana_mask){

  if(ana_mask & ADC0)
    Serial.println("ADC0");
  if(ana_mask & ADC1)
    Serial.println("ADC1");
  if(ana_mask & ADC2)
    Serial.println("ADC2");
```

```
  if(ana_mask & ADC3)
    Serial.println("ADC3");
  if(!ana_mask)
    Serial.println("No Analog IO");
}

// Logic to check which pins are enabled on remote XBee and pull
// relevant data
String read_Xbee(byte XBee_buf[30]){

static String xbee_print;
float time_s = 0;
byte analogMSB = 0, analogLSB = 0;
word analogReading = 0;
byte ADC0_MSB = 19; //default first analog data byte
byte ADC1_MSB = ADC0_MSB;
byte ADC2_MSB = ADC1_MSB;
byte ADC3_MSB = ADC2_MSB;
float temp = 0;
//float pressure = 48; // pressure dummy variable
word digi_mask = 0; // digital IO mask
byte ana_mask = 0; // analog (ADC) input mask
byte isDigital = 0;// is there digital input on the XBees?

  digi_mask = XBee_buf[17] | (XBee_buf[16] << 8); // digital mask
  ana_mask = XBee_buf[18]; // XBee analog channel mask
  isDigital = digi_check(digi_mask); //check which digital IO enabled
  ADC0_MSB = isDigital?21:19; //21 if digital pins, 19 otherwise
```

```
  ADC1_MSB = (ana_mask&ADC0)?(ADC0_MSB+2):ADC0_MSB;
  ADC2_MSB = (ana_mask&ADC1)?(ADC1_MSB+2):ADC1_MSB;
  ADC3_MSB = (ana_mask&ADC2)?(ADC2_MSB+2):ADC2_MSB;
  ana_check(ana_mask);

   if((XBee_buf[XBEE_ADDR_LSBy]==R1_LSBy)||
(XBee_buf[XBEE_ADDR_LSBy]==R2_LSBy)||(XBee_buf[XBEE_ADDR_LSBy]==R3_LSBy)||(XBee_buf[XBEE_ADDR_LSBy]==
R4_LSBy)){
    Serial.print(String(XBee_buf[XBEE_ADDR_LSBy],HEX) + ": ");
    time_s = (millis())/1000.0; // Time in seconds
    Serial.println(String(time_s, DEC) + " Seconds");
    if(ana_mask & ADC0){//if there is a pressure sensor
     analogMSB = XBee_buf[ADC0_MSB];
     analogLSB = XBee_buf[ADC0_MSB+1];
     analogReading = analogLSB | (analogMSB << 8);
     pressure = get_pressure(analogReading,'R');
    }
    if(ana_mask & ADC1){//if there is a temperature sensor
     analogMSB = XBee_buf[ADC1_MSB];
     analogLSB = XBee_buf[ADC1_MSB+1];
     analogReading = analogLSB | (analogMSB << 8);
     temp = getTemp_F_LM19(analogReading,'R'); // indicate remote data
    }
    if(ana_mask & ADC2){

    }
    if(ana_mask & ADC3){
```

```
    }
    xbee_print = String(String(millis(),DEC) + "," +
        String(XBee_buf[XBEE_ADDR_LSBy], HEX) + "," + String(temp, DEC)
         + "," + String(pressure,DEC));
   }
   else{
    Serial.println("Unknown Xbee");
    xbee_print = "UnknownXbee";
   }
   return xbee_print;
}

float get_pressure(word analogPres, char charlie){
 float tempPres = 0;
 if(charlie == 'R'){ // if remote data
  tempPres = analogPres*1.220/1023.0; // voltage at resistor divider output
  tempPres = tempPres * (R1_PRES_VAL + R2_PRES_VAL) / R1_PRES_VAL; // voltage at output
 }
 else if(charlie == 'L'){ // if local data
  tempPres = analogPres*3.3000/1023.0; //voltage measured at pressure output pin
 }
 tempPres = 18.75*((tempPres/V_Sup)-0.1);
 Serial.println(String(tempPres,DEC) + " PSIa");
 return tempPres;
}
```

```
/*
 * File:   simple_control.h
 * Author: richardpark
 *
 * Created on April 25, 2015, 4:51 PM
 */

#ifndef SIMPLE_CONTROL_H
 #define          SIMPLE_CONTROL_H


//*****************************************************************************************************
**********************
 //Last updated 5/2/2012
 //Josh's targeted control algorithm
 //These are used by simple_control(), steerToTarget(), landingRoutine(), and landingFlare()

 #include "Arduino.h"

 typedef struct servo
 {
          float value;
          float value_failsafe;

     int converted_value;
     //int converted_failsafe;
```

```
} servo;



//Global variables:

extern float headingAct[2]; //actual heading, initialize {0,0} here; updated in loop

extern float servoOutPrcnt[2]; //intialize {0,0}; corrective steering command to servo without amplification scaling, a fn of headingDev

extern float desiredYawRate[2]; //intialize {0,0}; minimum desired yaw rate, a fn of headingDev, degrees/s, must be LESS THAN 180 deg/s or steering routine WILL FAIL

extern int steeringActive; //intialize 0; 1 if steering occured previously this loop, 0 if not

extern float steeringScale[2]; //intialize {0,0}; servo scaling factor to achieve desired yaw

extern float achievedYawRate[2]; //intialize {0,0}; yaw rate achieved by previous turn command, degrees/s

extern int steerDelayCounter; //intialize 0; counter for initial steering input lag delay

extern int turnedLeft[2]; //intialize {0,0}; flag -- if true, system turned left last loop (but NOT as a result of negative servo value)

extern int turnedRight[2]; //intialize {0,0}; flag -- if true, system turned right last loop (but NOT as a result of negative servo value)

extern double headingTime[2]; //initialize {0,0}; holds GPS time-of-week (TOW) in milliseconds


// (time, lat, long, satellites, alt, velocity)

extern float gGPSheading;

extern float gGPSlatitude;

extern float gGPSlongitude;

extern float gGPSgndspeed;

extern float gGPSaltitude;

extern double gGPSTOW;


//Function prototypes:
```

```
void init_Servos(void);
void Servos_Update_All(void);
void simple_control(void);
void steerToTarget(float headingDes);
void landingRoutine(float headingDes);
void landingFlare(void);
void flight_control(String);

float convert_latitude_to_decimal_degrees(String latitude);
float convert_longitude_to_decimal_degrees(String longitude);



void test_servos(void);

//Landing target parameters:
//#define TARGETLAT 37.2862100 //target latitude in decimal degrees
//#define TARGETLONG -121.8517000 //target longitude in decimal degrees
//#define TARGET_ALTITUDE 72.1550 //elevation of landing target coordinates above sea level, meters

#define TARGETLAT 46.852814//target latitude in decimal degrees
#define TARGETLONG -117.619322 //target longitude in decimal degrees
#define TARGET_ALTITUDE 504.0000 //elevation of landing target coordinates above sea level, meters

//Distance and heading calculation constants:
#define EARTHRAD 6371000 //radius of the Earth, average, m

//Steering algorithm tuning parameters:
#define HEADING_DEADBAND 2.5 //deadband +/- degrees deviation between actual and desired; within this range no servo updates will occur this loop
```

#define DESIRED_YAW_DEADBAND 1 //yaw rate deadband +/- degrees/sec for no yaw rate adjustment in next loop (holds servo values constant)

#define NUM_LOOPS_BEFORE_SCALING_TURN 1 //number of loops to hold steering value before stepping to a new servo pull value; value of 1 updates every loop, 2 every 2 loops, etc.

#define FINE_SCALING_CUTOFF_DEG_SEC 2.5 //if yaw rate deviation from actual to desired (+/-) is less than this value, steering gain stepping uses fine increments

#define FINE_SCALING_STEP_PERCENT 0.010 //range 0 to 1; servo pull percent change each update (10 times/sec) to achieve target yaw rate when yaw rate deviation < FINE_SCALING_CUTOFF_DEG_SEC

#define FINE_SCALING_UNWIND_GAIN 1 //damping factor to increase rate of toggle line unwind; prevents underdamped overshoo of yaw rate and desired heading at low deviation angles

//usage example: 1.5 means toggle line unwrap step size will be 50% greater than toggle line pull step size (150% of pull rate); 1 is inactive

#define COARSE_SCALING_STEP_PERCENT 0.020 //range 0 to 1; servo pull percent change each update (10 times/sec) to achievetarget yaw rate when yaw rate deviation >= FINE_SCALING_CUTOFF_DEG_SEC

#define COARSE_SCALING_UNWIND_GAIN 1.5 //damping factor to increase rate of toggle line unwind; prevents underdamped overshoot of yaw rate and desired heading at low deviation angles

//usage example: 1.5 means toggle line unwrap step size will be 50% greater than toggle line pull step size (150% of pull rate)

//Desired yaw rate coefficients; controls desired yaw rate for a given heading deviation -- get these from Excel plot of esired yaw rate vs. heading deviation

#define DESIREDYAW_COEFF_1 -0.00000000000937 //x^6 term

#define DESIREDYAW_COEFF_2 0.00000000662071 //x^5 term

#define DESIREDYAW_COEFF_3 -0.00000185521159 //x^4 term

#define DESIREDYAW_COEFF_4 0.00026074766721 //x^3 term

#define DESIREDYAW_COEFF_5 -0.01906582376881 //x^2 term

#define DESIREDYAW_COEFF_6 0.76467370416140 //x term

#define DESIREDYAW_COEFF_7 -0.41502129438777 //constant term

//Landing routine constants:

```
  #define NUM_LOOPS_BEFORE_SCALING_TURN_LANDING 1 //number of loops to hold steering value before stepping to a new servo pull value; value of 1 updates every loop, 2 every 2 loops, etc.

  #define FINE_SCALING_CUTOFF_DEG_SEC_LANDING 5 //if yaw rate deviation (+/-) is less than this value, steering gain stepping uses fine increments

  #define FINE_SCALING_STEP_PERCENT_LANDING 0.015 //range 0 to 1; servo pull percent change each update (10 times/sec) to achieve target yaw rate when yaw rate deviation < FINE_SCALING_CUTOFF_DEG_SEC

  #define COARSE_SCALING_STEP_PERCENT_LANDING 0.035 //range 0 to 1; servo pull percent change each update (10 times/sec) toachieve target yaw rate when yaw rate deviation >= FINE_SCALING_CUTOFF_DEG_SEC

  #define LANDING_RADIUS_THRESHOLD 7.5 //when within this distance from target, landingRoutine is active, m

  #define LANDING_RADIUS 6 //desired spiral radius from target, m; MUST BE LESS THAN LANDING_RADIUS_THRESHOLD!!!

  //used to calculate required yaw rate to steer within LANDING_RADIUS_THRESHOLD

  //be careful with size! smaller size requires a greater yaw rate!


  //Flare routine constants:

  #define FLARE_HEIGHT 4 //distance above ground to initiate flare maneuver, meters

  #define FLARE_PRCNT 1.0 //range 0 to 1; percentage of maximum servo pull used in flare maneuver


  //END Josh's code

//*************************************************************************************************************


  //*****************************************************************//

  //***Debug Code Start -- comment ALL out for Monkey!***

  //

  //Used to feed fake data to control routine for compilation on

  //Windows-based PC


  #define DEGREES_TO_RAD 0.017453292519943295769236907684886
```

```
#define RAD_TO_DEG 57.295779513082320876798154814105

//#define SERVO_RIGHT_WINCH_3 2

//#define SERVO_LEFT_WINCH_4 3

#define SERVO_RIGHT_WINCH_3 0

#define SERVO_LEFT_WINCH_4 1

#define SERVO_R_WINCH_MAX_TRAVEL 5.75

#define SERVO_L_WINCH_MAX_TRAVEL -5.75

#define SERVO_R_WINCH_SCALE_FACTOR 1.75

#define SERVO_L_WINCH_SCALE_FACTOR -1.75


#define SCALE_FACTOR 3.0

#define MIN_TRAVEL 58.0 // 360 degrees CCW of center

#define CENTER_TRAVEL 90.0

#define MAX_TRAVEL 123.0 // 360 degrees CW of center

#define LEFT_SERVO 5

#define RIGHT_SERVO 6


//***Debug Code End***

//*********************************************************************//




#endif   /* SIMPLE_CONTROL_H */
```

```
/* ********************************************************************************
 * File Name:       XbeeDigital.h
 * Author:          Stephen Wayne
 * Date:            December 29, 2014
 * Revised:         April 27, 2015

 * Header Description:
 ********************************************************************************* */

#ifndef XBEEIO_H
 #define XBEEIO_H

 #include "Arduino.h"
 #include "MS5803_12BA.h"

 #define XBee_Serial Serial2
 #define XBEE_DATA_RATE 9600

 // Digital IO
 #define DIO0 0x01
 #define DIO1 0x02
 #define DIO2 0x04
 #define DIO3 0x08
 #define DIO4 0x10
 #define DIO5 0x20
 #define DIO6 0x40
```

```
#define DIO7 0x80
#define DIO10 0x400
#define DIO11 0x800
#define DIO12 0x1000

// Analog IO
#define ADC0 0x01
#define ADC1 0x02
#define ADC2 0x04
#define ADC3 0x08

#define R1_LSBy 0x04 //endpoint 1
#define R2_LSBy 0x16 // endpoint 2
#define R3_LSBy 0xA4 //endpoint 3
#define R4_LSBy 0x0F // Coordinator
#define XBEE_ADDR_LSBy 11 //least significant byte

#define R1_PRES_VAL 50500 // voltage divider values, in ohms
#define R2_PRES_VAL 98220

#define V_Sup 3.3
// Function Prototypes
void init_XBee(void);
byte digi_check(word);
void ana_check(byte);
String read_Xbee(byte XBee_buf[30]);
float get_pressure(word,char);
// End of XbeeIO.h
```

```
#endif
#include <Servo.h>

//*********************************************************************//
//***Debug Code Start -- comment ALL out for Monkey!***
//This feeds fake data to the control routines for compilation, simulation,
//and debugging on a Windows PC
#include <math.h>
#include <stdio.h>
#include "simple_control.h"
#include <Servo.h>
#include <stdlib.h>


//GLOBALS
//***Debug Code End***

//*************************************************************************************
//Start Josh's control code -- current version 3/12/2012 -- last updated 4/22/2012 44
//*********************************************************************//
//Function: simple_control
//
//Desc: Control loop for flight; determines heading and distance to target
//Receives:
//Returns:
//CONSTANTS: EARTHRAD, TARGETLAT, TARGETLONG, LANDING_RADIUS_THRESHOLD,
// DEGREES_TO_RAD, RAD_TO_DEG, TARGET_ALTITUDE, FLARE_HEIGHT
```

```
//Globals: headingAct[2]


//Global variables:

float headingAct[2] = {0,0}; //actual heading, initialize {0,0} here; updated in loop

float servoOutPrcnt[2] = {0,0}; //intialize {0,0}; corrective steering command to servo without amplification scaling, a fn of headingDev

float desiredYawRate[2] = {0,0}; //intialize {0,0}; minimum desired yaw rate, a fn of headingDev, degrees/s, must be L ESS THAN 180 deg/s or steering routine WILL FAIL

int steeringActive = 0; //intialize 0; 1 if steering occured previously this loop, 0 if not

float steeringScale[2] = {0,0}; //intialize {0,0}; servo scaling factor to achieve desired yaw

float achievedYawRate[2] = {0.0}; //intialize {0,0}; yaw rate achieved by previous turn command, degrees/s

int steerDelayCounter = 0; //intialize 0; counter for initial steering input lag delay

int turnedLeft[2] = {0,0}; //intialize {0,0}; flag -- if true, system turned left last loop (but NOT as a result of negative servo value)

int turnedRight[2] = {0,0}; //intialize {0,0}; flag -- if true, system turned right last loop (but NOT as a result of negative servo value)

double headingTime[2] = {0,0}; //initialize {0,0}; holds GPS time-of-week (TOW) in milliseconds

float gGPSheading = 0;//284;

float gGPSlatitude = 0.0;//37.2862000;

float gGPSlongitude = 0.0;//-121.8517000;

float gGPSgndspeed = 0;//15;

float gGPSaltitude = 0;//82.1550;

double gGPSTOW = 0;//579857000;


servo servo_out[2];


Servo servoL;

Servo servoR;


void init_Servos(void){
```

```
  servoL.attach(LEFT_SERVO);
  servoR.attach(RIGHT_SERVO);
  servoL.write(CENTER_TRAVEL);
  servoR.write(CENTER_TRAVEL); // 2 is minimum, via testing
  delay(1000);

}

void Servos_Update_All(void){
  float leftVal = ((servo_out[SERVO_LEFT_WINCH_4].value * 90.0)/SCALE_FACTOR);
  float rightVal = ((servo_out[SERVO_RIGHT_WINCH_3].value * 90.0)/SCALE_FACTOR);

  leftVal = leftVal/10.0;
  rightVal = rightVal/10.0; // scale properly

  leftVal = CENTER_TRAVEL + leftVal;
  rightVal = CENTER_TRAVEL + rightVal;

  // Very ugly logic to follow:
  if(leftVal >= MAX_TRAVEL) servo_out[SERVO_LEFT_WINCH_4].converted_value = MAX_TRAVEL;
  else if(leftVal <= MIN_TRAVEL) servo_out[SERVO_LEFT_WINCH_4].converted_value = MIN_TRAVEL;
  else if(leftVal < CENTER_TRAVEL) servo_out[SERVO_LEFT_WINCH_4].converted_value = (int)floor(leftVal);
  else servo_out[SERVO_LEFT_WINCH_4].converted_value = (int)ceil(leftVal);
  if(rightVal >= MAX_TRAVEL) servo_out[SERVO_RIGHT_WINCH_3].converted_value = MAX_TRAVEL;
  else if(rightVal <= MIN_TRAVEL) servo_out[SERVO_RIGHT_WINCH_3].converted_value = MIN_TRAVEL;
  else if(rightVal < CENTER_TRAVEL) servo_out[SERVO_RIGHT_WINCH_3].converted_value = (int)floor(rightVal);
  else servo_out[SERVO_RIGHT_WINCH_3].converted_value = (int)ceil(rightVal);
```

```
  Serial.println(servo_out[SERVO_LEFT_WINCH_4].value);
  Serial.println(servo_out[SERVO_LEFT_WINCH_4].converted_value);
  Serial.println(servo_out[SERVO_RIGHT_WINCH_3].value);
  Serial.println(servo_out[SERVO_RIGHT_WINCH_3].converted_value);

  servoL.write(servo_out[SERVO_LEFT_WINCH_4].converted_value);
  servoR.write(servo_out[SERVO_RIGHT_WINCH_3].converted_value);
}

void test_servos(void){

  int pos = 0;

   for(pos = 0; pos <= 180; pos += 1) // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    servoL.write(pos);               // tell servo to go to position in variable 'pos'
    servoR.write(pos);
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for(pos = 180; pos>=0; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    servoL.write(pos);               // tell servo to go to position in variable 'pos'
    servoR.write(pos);
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}

void simple_control(void)
```

```
{
  // Local variables:
  // (for arrays, 0 is current value and 1 is value from previous loop)
  // current device latitude, dec. degrees
  double latCurrent;
  // current device longitude, dec. degrees
  double longCurrent;
  // current device altitude AGL, meters
  double altCurrent;
  // difference between target and current latitude
  double dLat;
  // difference between target and current longitude
  double dLong;
  // current pos. latitude in radians
  double latCurrentRad;
  // target pos. latitude in radians
  double targetLatRad;
  // value for Haversine distance calculation
  double aDist;
  // value for Haversine distance calculation
  double cDist;
  // variable for heading-to-target calculation
  // 'yaw
  double yHead;
  // variable for heading-to-target calculation
  double xHead;
  // desired heading to target, calculated at each loop update, degrees
  float headingDes;
```

```
  // distance magnitude from current pos. to target, m
  float distMag;
  // holds rounded desired heading to work with modulo operator
  int headingDesRnd;
  //DEBUG LINE
//  printf("x________\nStart control
loop\n************************************************************\n");


  //printf("\n_____\n");
  //printf("Start control loop\n************************************************************\n");


  //Writes current lat/long in degrees from Monkey
  //latCurrent = gGPS.latitude;
  //longCurrent = gGPS.longitude;
  //altCurrent = gGPS.altitude;
  //DEBUG LINE

  latCurrent = gGPSlatitude;
  longCurrent = gGPSlongitude;
  altCurrent = gGPSaltitude;

  //Calculates distMag using Haversine formula
  dLat = ( TARGETLAT - latCurrent ) * DEGREES_TO_RAD;
  dLong = ( TARGETLONG - longCurrent ) * DEGREES_TO_RAD;

  //current latitude in radians
  latCurrentRad = latCurrent * DEGREES_TO_RAD;

  //current longitude in radians
```

```
targetLatRad = TARGETLAT*DEGREES_TO_RAD;

aDist = sin( dLat / 2 )     * sin( dLat / 2 )
  + sin( dLong / 2 )* sin( dLong / 2 )
  * cos( latCurrentRad ) * cos( targetLatRad );

cDist = 2 * atan2( sqrt ( aDist ), sqrt( 1 - aDist ));

distMag = EARTHRAD * cDist;

//Calculates headingDes using results from above
yHead = sin( dLong ) * cos( targetLatRad );

xHead = cos( latCurrentRad ) * sin( targetLatRad )
  - sin( latCurrentRad )
  * cos(targetLatRad)*cos(dLong);

//returns desired heading in degrees from -180 to 180
headingDes = atan2( yHead, xHead ) * RAD_TO_DEG;

//rounds headingDes for the module oeprator on next line for the modulo operator on next line
headingDes >= 0 ? ( headingDesRnd = (int)( headingDes+0.5 ))
  : ( headingDesRnd = (int)( headingDes-0.5 ));

//uses headingDesRnd to return 0 <= int headingDes < 360
headingDes = (headingDesRnd+360) % 360;

//DEBUG LINE
```

```
//  printf("\n Distance to target: %.3f\n Desired heading: %.3f\n Altitude: %.3f\n", distMag, headingDes, altCurrent);


  //printf("Distance to target: %.3f\n", distMag);
  //printf("Desired heading: %.3f\n", headingDes);
  //printf("Altitude: %.3f\n", altCurrent);


  // If within desired radius of target, start circling
  if (altCurrent < (TARGET_ALTITUDE + FLARE_HEIGHT))
    {
      landingFlare();
    }


  // If not to target yet, keep flying to it
  else if (distMag < LANDING_RADIUS_THRESHOLD)
    {
      landingRoutine(headingDes);
    }
  else
    {
      steerToTarget ( headingDes );
    }
}
//End control loop routine



//*****************************************************************//
//Function: steerToTarget
```

```
//
//Desc: Steers device proportionally as-the-crow-flies to target coords.
//
//Receives: float headingDes
//
//Returns:
//
//CONSTANTS: HEADING_DEADBAND, DESIREDYAW_COEFF[1-7], NUM_LOOPS_BEFORE_SCALING_TURN,
// DESIRED_YAW_DEADBAND, SERVO_RIGHT_WINCH_3, SERVO_LEFT_WINCH_4,
// SERVO_R_WINCH_MAX_TRAVEL, SERVO_L_WINCH_MAX_TRAVEL, SERVO_R_WINCH_SCALE_FACTOR,
// SERVO_L_WINCH_SCALE_FACTOR, FINE_SCALING_UNWIND_GAIN, COARSE_SCALING_UNWIND_GAIN,
// FINE_SCALING_STEP_PERCENT, COARSE_SCALING_STEP_PERCENT, FINE_SCALING_CUTOFF_DEG_SEC
//
//Globals: headingAct[2], servoOutPrcnt[2], desiredYawRate[2], steeringActive,
// steeringScale, achievedYawRate[2], steerDelayCounter, turnedLeft[2], turnedRight[2],
// gGPS.heading, gGPS.latitude, gGPS.longitude, gGPS.TOW, headingTime[2]
//_____________________________________________________________________________//

void steerToTarget(float headingDes)
{
  //Local variables:

  //deviation angle from actual heading to desired, degrees
  float headingDev;

  //loop counter for rewriting achievedYawRate array values to "older" positions in array
  int loopCtr;
```

```
  //DEBUG LINE - comment out below for debug
  //Led_On(LED_RED); //Red LED blinks while control is in this routine (turned off at end of loop)

  //Writes current GPS heading in degrees and GPS time-of-week in milliseconds from Monkey to arrays' 0 position
  //headingAct[0] = gGPS.heading;
  //headingTime[0] = gGPS.TOW; //NOTE: using TOW for headingTime will cause a momentary glitch when TOW reverts to 0 each week

  //DEBUG LINE
  headingAct[0] = gGPSheading;
  headingTime[0] = gGPSTOW;

  //DEBUG LINES BELOW
  //printf("Actual heading: %.3f\n", headingAct[0]);
  //printf("***Start steer to target routine***\n");

  //Determines the flight heading deviation from actual to the desired heading
  //Positive clockwise, negative counterclockwise; range -180 to 180 degrees
  if ((headingDes-headingAct[0] >= -180) && (headingDes-headingAct[0] <= 180))
   {
     headingDev = headingDes-headingAct[0];
   }
  else if (headingDes-headingAct[0] < -180)
   {
     headingDev = 360 + (headingDes-headingAct[0]);
   }
  else
   {
     headingDev = (headingDes-headingAct[0]) - 360;
```

```
 }

//Determines heading angle (yaw) change from previous loop to current loop and achievedYawRate in deg/s
//Positive clockwise, negative counterclockwise
//Delivers rate achieved between -180 to 180 deg/s

//only updates achievedYawRate when a new GPS heading is available; approx. 4 times/s ec
if ( headingAct[0] != headingAct[1] )
 {
   if (( headingAct[0] - headingAct[1] >= -180 ) && ( headingAct[0] - headingAct[1] <= 180 ))
         {
           achievedYawRate[0] = (360 + headingAct[0]-headingAct[1])/((headingTime[0]-headingTime[1])/1000);
         }
   else if (headingAct[0]-headingAct[1] < -180)
         {
           achievedYawRate[0] = (headingAct[0]-headingAct[1] - 360)/((headingTime[0]-headingTime[1])/1000);
         }
   else
         {
           achievedYawRate[0] = (headingAct[0]-headingAct[1])/((headingTime[0]-headingTime[1])/1000);
         }
 }


//Returns desiredYawRate = fn(headingDev) as a rate (+/- deg/s), positive clockwise
if (headingDev<0)
 {
```

```
    desiredYawRate[0] = ( pow( fabs( headingDev ), 6 ) * DESIREDYAW_COEFF_1
                        + pow( fabs( headingDev ), 5 ) * DESIREDYAW_COEFF_2
                        + pow( fabs( headingDev ), 4 ) * DESIREDYAW_COEFF_3
                        + pow( fabs( headingDev ), 3 ) * DESIREDYAW_COEFF_4
                        + pow( fabs( headingDev ), 2 ) * DESIREDYAW_COEFF_5
                        + ( fabs( headingDev )) * DESIREDYAW_COEFF_6
                        + DESIREDYAW_COEFF_7 ) * (-1);
  }
 else
  {
    desiredYawRate[0] = ( pow( fabs( headingDev ), 6 ) * DESIREDYAW_COEFF_1
                        + pow( fabs( headingDev ), 5 ) * DESIREDYAW_COEFF_2
                        + pow( fabs( headingDev ), 4 ) * DESIREDYAW_COEFF_3
                        + pow( fabs( headingDev ), 3 ) * DESIREDYAW_COEFF_4
                        + pow( fabs( headingDev ), 2 ) *DESIREDYAW_COEFF_5
                        + ( fabs( headingDev )) * DESIREDYAW_COEFF_6
                        + DESIREDYAW_COEFF_7);
  }

 // Writes steering scale value of previous loop to "old" array value
 steeringScale[1] = steeringScale[0];

 // Resets steeringScale to starting value when turn direction changes as a result of crossing over desired heading,
 // or in heading deadband to prepare for next turn AFTER leaving deadband
 if ( !steeringActive || (turnedLeft[0] == 1 && headingDev > HEADING_DEADBAND) || (turnedRight[0] == 1 &&
headingDev < -HEADING_DEADBAND))
  {
    steeringScale[0] = 0;
    steeringScale[1] = 0;
```

```
  }


 // Writes previous turn flag values to "old" array values

 turnedLeft[1] = turnedLeft[0];

 turnedRight[1] = turnedRight[0];


 // Calculates new steeringScale value based on desired yaw rate and achieved delta yaw angle during previous control loop

 // If yaw rate is within deadband, or enough loops haven't occurred before update, steeringScale[0] is unchanged

 // If NUM_LOOPS_BEFORE_SCALING_TURN = 1, steering value is changed every loop; if = 2, changed every 2 loops, and so on

 if (steerDelayCounter % NUM_LOOPS_BEFORE_SCALING_TURN == 0)

  {

   // Don't scale unless +/- yaw rate error is greater than deadband value AND we are not in the heading deadband

   if((fabs(achievedYawRate[0]-desiredYawRate[0]) > DESIRED_YAW_DEADBAND) && (fabs(headingDev) > HEADING_DEADBAND))

         {

           //In the case of different signs for desired yaw rate vs. achieved yaw rate, increase servo pull:

           //For yaw rate deviation less than cutoff threshold, use FINE scaling

           if((achievedYawRate[0] / desiredYawRate[0] < 0) && (fabs(desiredYawRate[0]-achievedYawRate[0]) < FINE_SCALING_CUTOFF_DEG_SEC))

            {

              steeringScale[0] = steeringScale[1] + FINE_SCALING_STEP_PERCENT;

            }

           //For yaw rate deviation greater than cutoff threshold, use COARSE scaling

           else if((achievedYawRate[0] / desiredYawRate[0] < 0) && (fabs(desiredYawRate[0]-achievedYawRate[0]) >= FINE_SCALING_CUTOFF_DEG_SEC))

            {

              steeringScale[0] = steeringScale[1] + COARSE_SCALING_STEP_PERCENT;

            }
```

```
//For yaw rate deviation less than cutoff threshold, use FINE scaling
else if(fabs(desiredYawRate[0]-achievedYawRate[0]) < FINE_SCALING_CUTOFF_DEG_SEC)
 {
   //Increase yaw rate, fine stepping
   if((achievedYawRate[0] >= 0 && desiredYawRate[0] >= 0) && (achievedYawRate[0] < desiredYawRate[0]))
         {
          steeringScale[0] = steeringScale[1] + FINE_SCALING_STEP_PERCENT;
         }
   //Increase yaw rate, fine stepping
   else if((achievedYawRate[0] < 0 && desiredYawRate[0] < 0) && (achievedYawRate[0] >= desiredYawRate[0]))
         {
          steeringScale[0] = steeringScale[1] + FINE_SCALING_STEP_PERCENT;
         }
   else
         {
          //Decrease yaw rate, fine stepping
          steeringScale[0] = steeringScale[1] - (FINE_SCALING_STEP_PERCENT * FINE_SCALING_UNWIND_GAIN);
         }
 }
//For yaw rate deviation more than cutoff threshold, use COARSE scaling
else if(fabs(desiredYawRate[0]-achievedYawRate[0]) >= FINE_SCALING_CUTOFF_DEG_SEC)
 {
   //Increase yaw rate, coarse stepping
   if((achievedYawRate[0] >= 0 && desiredYawRate[0] >= 0) && (achievedYawRate[0] < desiredYawRate[0]))
         {
          steeringScale[0] = steeringScale[1] + COARSE_SCALING_STEP_PERCENT;
         }
   //Increase yaw rate, coarse stepping
```

```
            else if((achievedYawRate[0] < 0 && desiredYawRate[0] < 0) && (achievedYawRate[0] >= desiredYawRate[0]))
                  {
                    steeringScale[0] = steeringScale[1] + COARSE_SCALING_STEP_PERCENT;
                  }
            //Decrease yaw rate, coarse stepping
            else
                  {
                    steeringScale[0] = steeringScale[1] - (COARSE_SCALING_STEP_PERCENT *
COARSE_SCALING_UNWIND_GAIN);
                  }
          }
        }
   }

 //Keeps the system from driving servos past 100% pull
 if (steeringScale[0] > 1)
  {
    steeringScale[0] = 1;
  }

 if (steeringScale[0] < -1)
 {
   steeringScale[0] = -1;
 }

 //  ????? where does this go ???? : In the case of desired and achieved yaw rate both positive or both negative:

 //servoOutPrcnt is the fraction of maximum servo pull for turning, 0 to 1
```

```
 servoOutPrcnt[0] = steeringScale[0];


 //DEBUG LINE
 //printf("\nThis loop's steering commands:\n steeringScale[0,1]: [%.6f, %6f]\n servoOutPrcnt[0,1]: [%.6f, %.6f]\n",
        //          steeringScale[0], steeringScale[1], servoOutPrcnt[0], servoOutPrcnt[1]);


 //Writes current pre-turn values to previous values for next loop
 servoOutPrcnt[1] = servoOutPrcnt[0];


 //Determines which direction to turn and commands the servos to steer, scaled by steeringScale
 //If steeringScale=1, full deflection is commanded (5.75 inch toggle line pull on parafoil)
 if ((headingDev < -HEADING_DEADBAND && servoOutPrcnt[0] >= 0) || (headingDev > HEADING_DEADBAND &&
servoOutPrcnt[0] < 0 ))
  {
    //then turn left!
    servo_out[ SERVO_LEFT_WINCH_4 ].value = servo_out[ SERVO_LEFT_WINCH_4 ].value_failsafe
        - ( fabs( servoOutPrcnt[0] ) * SERVO_L_WINCH_MAX_TRAVEL * SERVO_L_WINCH_SCALE_FACTOR );


      //printf("servo_out: %f", servo_out[ SERVO_LEFT_WINCH_4 ].value);


    servo_out[ SERVO_RIGHT_WINCH_3 ].value = servo_out[ SERVO_RIGHT_WINCH_3 ].value_failsafe;
    //For DEBUG, disable next line
    Servos_Update_All();


    // flag set for left turn, but ONLY for a turn not caused negative servo value
    if (headingDev < -HEADING_DEADBAND && servoOutPrcnt[0] >= 0)
        {
          turnedLeft[0] = 1;
          turnedRight[0] = 0;
```

```
        }
    //DEBUG LINE BELOW
    //printf("\nSteering Left\n");
   }
  //then turn right!
  else if ((headingDev > HEADING_DEADBAND && servoOutPrcnt[0] >= 0) || (headingDev < -HEADING_DEADBAND &&
servoOutPrcnt[0 ] < 0))
   {
    servo_out[ SERVO_RIGHT_WINCH_3 ].value = servo_out[ SERVO_RIGHT_WINCH_3 ].value_failsafe
        + (fabs(servoOutPrcnt[0])*SERVO_R_WINCH_MAX_TRAVEL*SERVO_R_WINCH_SCALE_FACTOR);



    servo_out[ SERVO_LEFT_WINCH_4 ].value = servo_out[ SERVO_LEFT_WINCH_4 ].value_failsafe;
    //For DEBUG, disable next line
    Servos_Update_All();

    // flag set for left turn, but ONLY for a turn not caused by negative servo value
    if (headingDev > HEADING_DEADBAND && servoOutPrcnt[0] >= 0)
        {
          turnedRight[0] = 1;
          turnedLeft[0] = 0;
        }

    //DEBUG LINE BELOW
    //printf("\nSteering Right\n");
   }
  else
   {
    turnedLeft[0] = 0;
```

```
    turnedRight[0] = 0;


    //DEBUG LINE BELOW
    //printf("\nServos Unchanged -- In Heading Deadband\n");
   }

//        } // ************** ADDED MISSING CLOSE PARENS? ************************//



  //Sets steeringActive flag true if steering occurred earlier this loop and
  //disables the reset of steering scale to 1 until straight flight resumes
  if (headingDev > -HEADING_DEADBAND && headingDev < HEADING_DEADBAND)
   {
    steeringActive = 0;
   }
  else
   {
    steeringActive = 1;
   }



    //Counter for how many loops occur at minimum steer before steer output begins scaling //Intended to account for delay in heading change after initial steering input
  if (steeringActive)
   {
    steerDelayCounter+= 1;
   }
  else
   {
```

```
    steerDelayCounter = 0;
  }



 //DEBUG LINE
 //printf("\nValues for this loop:\n steerDelayCounter (value for next loop): %d\n headingDev from actual-->desired:
%.3f\n"
        //          " achievedYawRate[0] (prev. loop to now): %.3f\n"
        //          " desiredYawRate[0,1]: [%.3f, %.3f]\n steeringActive: %d\n",
        //          steerDelayCounter, headingDev, achievedYawRate[0], desiredYawRate[0], desiredYawRate[1],
steeringActive );

 //Writes current values to previous values for next loop
 desiredYawRate[1] = desiredYawRate[0];
 headingAct[1] = headingAct[0];
 headingTime[1] = headingTime[0];
 achievedYawRate[1] = achievedYawRate[0];

 //DEBUG LINE - comment out below for debug
 //Led_Off(LED_RED); //Red LED blinks while control is in this routine (turned on at beginning of loop)
}



//*****************************************************************//
//Function: landingRoutine
//
//Desc: Steers device proportionally in a circle around the target coordinate,
// with radius <= LANDING_RADIUS_THRESHOLD
//
```

```
//Receives: float headingDes
//
//Returns:
//
//CONSTANTS: LANDING_RADIUS_THRESHOLD, HEADING_DEADBAND,
// NUM_LOOPS_BEFORE_SCALING_TURN_LANDING, DESIRED_YAW_DEADBAND,
// LANDING_RADIUS, DEGREES_TO_RAD, SERVO_RIGHT_WINCH_3, SERVO_LEFT_WINCH_4,
// SERVO_R_WINCH_MAX_TRAVEL, SERVO_L_WINCH_MAX_TRAVEL, SERVO_R_WINCH_SCALE_FACTOR,
// SERVO_L_WINCH_SCALE_FACTOR, FINE_SCALING_UNWIND_GAIN, COARSE_SCALING_UNWIND_GAIN
//
//Globals: headingAct[2], servoOutPrcnt[2], desiredYawRate[2], steeringActive,
// steeringScale, achievedYawRate[2], steerDelayCounter, turnedLeft[2], turnedRight[2],
// gGPS.heading, gGPS.latitude, gGPS.longitude, gGPS.gndspeed, gGPS.TOW, headingTime[2]
//_____//

void landingRoutine(float headingDes)
{

  //Local variables:
  //deviation angle from actual heading to desired, degrees
  float headingDev;
  //loop counter for rewriting achievedYawRate array values to "older" positions in array
  int loopCtr;

  //DEBUG LINE - comment out below for debug
  //Led_On(LED_RED); //Red LED is on steady in this routine
```

```
 //Writes current GPS heading in degrees and GPS time-of-week in milliseconds from Monkey to arrays' 0 position
 //headingAct[0] = gGPS.heading;
 //headingTime[0] = gGPS.TOW; //NOTE: using TOW for headingTime will cause a momentary glitch when TOW reverts to 0 each week
 //DEBUG LINE
 headingAct[0] = gGPSheading;
 headingTime[0] = gGPSTOW;

 //DEBUG LINES BELOW
//printf(" Actual heading: %.3f\n", headingAct[0]);
//printf("\n\n***Start landing routine***\n");

 //Determines the flight heading deviation from actual to the desired heading assuming straight flight to determine required landing spiral direction
 //Positive clockwise, negative counterclockwise; range -180 to 180 degrees
 if ((headingDes-headingAct[0] >= -180) && (headingDes-headingAct[0] <= 180))
  {
   headingDev = headingDes-headingAct[0];
  }
 else if (headingDes-headingAct[0] < -180)
  {
   headingDev = 360 + (headingDes-headingAct[0]);
  }
 else
  {
   headingDev = (headingDes-headingAct[0]) - 360;
  }
```

```
 //Determines new desired heading tangent to a radius around target, and the direction of spiral depending on target being approached from right or left side


 //If true, we are approaching target to the right, and should spiral in a left/CCW pattern
 if (headingDev < 0)
  {
    //Causes parafoil to fly tangent to a CCW circle around target
    headingDes = headingDes + 90;
  }
 //If true, we are approaching target to the left, and should fly in a right/CW pattern
 else if (headingDev >= 0)
  {
    headingDes = headingDes - 90;
  }



 //Now re-determine headingDev for the new spiral-flight desired heading
 //Determines the flight heading deviation from actual to the desired heading for landing spiral flight path
 //Positive clockwise, negative counterclockwise; range -180 to 180 degrees
 if ((headingDes-headingAct[0] >= -180) && (headingDes-headingAct[0] <= 180))
  {
    headingDev = headingDes-headingAct[0];
  }


 else if (headingDes-headingAct[0] < -180)
  {
    headingDev = 360 + (headingDes-headingAct[0]);
  }
 else
```

```
  {
    headingDev = (headingDes-headingAct[0]) - 360;
  }



//DEBUG LINE
//printf("\n Desired heading for landing spiral: %.3f\n", headingDes);


//Determines heading angle (yaw) change from previous loop to current loop and achievedYawRate in deg/s
//Positive clockwise, negative counterclockwise
//Delivers rate achieved between -180 to 180 deg/s
//only updates achievedYawRate when a new GPS heading is available; approx. 4 times/s ec
if (headingAct[0] != headingAct[1])
 {

   if ((headingAct[0]-headingAct[1] >= -180) && (headingAct[0]-headingAct[1] <= 180))
    {
          achievedYawRate[0] = (headingAct[0]-headingAct[1])/((headingTime[0]-headingTime[1])/1000);
    }
   else if (headingAct[0]-headingAct[1] < -180)
    {
          achievedYawRate[0] = (360 + headingAct[0]-headingAct[1])/((headingTime[0]-headingTime[1])/1000);
    }
   else
    {
          achievedYawRate[0] = (headingAct[0]-headingAct[1] - 360)/((headingTime[0]-headingTime[1])/1000);
    }
 }
```

```
//Returns desiredYawRate = fn(headingDev, LANDING_RADIUS, gGPS.gndspeed) as a rate (+/- deg/s), positive clockwise
//Calculates the yaw rate required to maintain a circular flight path of radius LANDING_RADIUS



//desiredYawRate[0]= (headingDev/fabs(headingDev) * gGPS.gndspeed / LANDING_RADIUS) * RAD_TO_DEG;
//DEBUG LINE -- make sure to activate above line for Monkey!
desiredYawRate[0]= (headingDev/fabs(headingDev) * gGPSgndspeed / LANDING_RADIUS) * RAD_TO_DEG;

//Prevents desired yaw rate from exceeding 180 deg/s, plus some margin. Steering will fail if actual
//yaw rate exceeds 180 deg/s.
if (desiredYawRate[0] > 170)
  {
    desiredYawRate[0] = 170;
  }
else if (desiredYawRate[0] < -170)
  {
    desiredYawRate[0] = -170;
  }

//Writes steering scale value of previous loop to "old" array value
steeringScale[1] = steeringScale[0];

// Resets steeringScale to starting value when turn direction changes as a result of crossing over desired heading,
// or in heading deadband to prepare for next turn AFTER leaving deadband
if (!steeringActive || (turnedLeft[0] == 1 && headingDev > HEADING_DEADBAND) || (turnedRight[0] == 1 &&
headingDev < -HEADING_DEADBAND))
  {
    steeringScale[0] = 0;
```

```
    steeringScale[1] = 0;
  }

 //Writes previous turn flag values to "old" array values
 turnedLeft[1] = turnedLeft[0];
 turnedRight[1] = turnedRight[0];

 //Calculates new steeringScale valfue based on desired yaw rate and achieved delta yaw angle during previous control loop
 //If yaw rate is within deadband, or enough loops haven't occurred before update, steeringScale[0] is unchanged

 //If NUM_LOOPS_BEFORE_SCALING_TURN_LANDING = 1, ste ering value is changed every loop; if = 2, changed every 2 loops, and so on
 if (steerDelayCounter % NUM_LOOPS_BEFORE_SCALING_TURN_LANDING == 0)
  {
    //Don't scale unless +/- yaw rate error is greater than deadband value AND we are not in the heading deadband
    if((fabs(achievedYawRate[0]-desiredYawRate[0]) > DESIRED_YAW_DEADBAND) && (fabs(headingDev) > HEADING_DEADBAND))
     {
          //In the case of different signs for desired yaw rate vs. achieved yaw rate, increase servo pull:
          //For yaw rate deviation less than cutoff threshold, use FINE scaling
          if((achievedYawRate[0] / desiredYawRate[0] < 0) && (fabs(desiredYawRate[0]-achievedYawRate[0]) < FINE_SCALING_CUTOFF_DEG_SEC_LANDING))
           {
             steeringScale[0] = steeringScale[1] + FINE_SCALING_STEP_PERCENT_LANDING;
           }
          //For yaw rate deviation greater than cutoff threshold, use coarse scaling
          else if((achievedYawRate[0] / desiredYawRate[0] < 0) && (fabs(desiredYawRate[0]-achievedYawRate[0]) >= FINE_SCALING_CUTOFF_DEG_SEC_LANDING))
           {
```

```
    steeringScale[0] = steeringScale[1] + COARSE_SCALING_STEP_PERCENT_LANDING;
   }
  //In the case of desired and achieved yaw rate both positive or both negative:
  //For yaw rate deviation less than cutoff threshold, use FINE scaling
  else if(fabs(desiredYawRate[0]-achievedYawRate[0]) < FINE_SCALING_CUTOFF_DEG_SEC_LANDING)
   {
    //Increase yaw rate, fine stepping
    if((achievedYawRate[0] >= 0 && desiredYawRate[0] >= 0) && (achievedYawRate[0] < desiredYawRate[0]))
     {
          steeringScale[0] = steeringScale[1] + FINE_SCALING_STEP_PERCENT_LANDING;
     }
    //Increase yaw rate, fine stepping
    else if((achievedYawRate[0] < 0 && desiredYawRate[0] < 0) && (achievedYawRate[0] >= desiredYawRate[0]))
     {
          steeringScale[0] = steeringScale[1] + FINE_SCALING_STEP_PERCENT_LANDING;
     }
    //Decrease yaw rate, fine stepping
    else
     {
          steeringScale[0] = steeringScale[1] - (FINE_SCALING_STEP_PERCENT_LANDING *
FINE_SCALING_UNWIND_GAIN);
     }
   }
  //For yaw rate deviat ion more than cutoff threshold, use COARSE scaling
  else if(fabs(desiredYawRate[0]-achievedYawRate[0]) >= FINE_SCALING_CUTOFF_DEG_SEC_LANDING)
   {
    //Increase yaw rate, coarse stepping
    if((achievedYawRate[0] >= 0 && desiredYawRate[0] >= 0) && (achievedYawRate[0] < desiredYawRate[0]))
     {
```

```
                steeringScale[0] = steeringScale[1] + COARSE_SCALING_STEP_PERCENT_LANDING;
            }
            //Increase yaw rate, coarse stepping
            else if((achievedYawRate[0] < 0 && desiredYawRate[0] < 0) && (achievedYawRate[0] >= desiredYawRate[0]))
            {
                steeringScale[0] = steeringScale[1] + COARSE_SCALING_STEP_PERCENT_LANDING;
            }
            else
            {
                //Decrease yaw rate, coarse stepping
                steeringScale[0] = steeringScale[1] - (COARSE_SCALING_STEP_PERCENT_LANDING *
COARSE_SCALING_UNWIND_GAIN);
            }
          }
        }
      }

//Keeps the system from driving servos past 100% pull
if (steeringScale[0] > 1)
 {
  steeringScale[0] = 1;
 }

if (steeringScale[0] < -1)
 {
  steeringScale[0] = -1;
 }

//servoOutPrcnt is the fraction of maximum servo pull for turning, 0 to 1
```

```
 servoOutPrcnt[0]=steeringScale[0];



 //DEBUG LINE

 //printf("\nThis loop's steering commands:\n steeringScale[0,1]: [%.6f, %6f]\n servoOutPrcnt[0,1]: [%.6f, %.6f]\n",
steeringScale[0], steeringScale[1], servoOutPrcnt[0], servoOutPrcnt[1]);


 //Writes current pre-turn values to previous values for next loop

 servoOutPrcnt[1] = servoOutPrcnt[0];




 //Determines which direction to turn and commands the servos to steer, scaled by steeringScale

 //If steeringScale=1, full deflection is commanded (5.75 inch toggle line pull on parafoil)


 //then turn left!

 if ((headingDev < -HEADING_DEADBAND && servoOutPrcnt[0] >= 0) || (headingDev > HEADING_DEADBAND &&
servoOutPrcnt[0] < 0 ))


  {

    servo_out[ SERVO_LEFT_WINCH_4 ].value = servo_out[ SERVO_LEFT_WINCH_4 ].value_failsafe

      - (fabs(servoOutPrcnt[0])*SERVO_L_WINCH_MAX_TRAVEL*SERVO_L_WINCH_SCALE_FACTOR);


    servo_out[ SERVO_RIGHT_WINCH_3 ].value = servo_out[ SERVO_RIGHT_WINCH_3 ].value_failsafe;

    //For DEBUG, disable next line

    Servos_Update_All();


    if (headingDev < -HEADING_DEADBAND && servoOutPrcnt[0] >= 0)

      //flag set for left turn, but ONLY for a turn not cause d by negative servo value
```

```
    {


        turnedLeft[0] = 1;

        turnedRight[0] = 0;

    }


   //DEBUG LINE BELOW

   //printf("\nSteering Left\n");

  }


 //then turn right!

 else if ((headingDev > HEADING_DEADBAND && servoOutPrcnt[0] >= 0) || (headingDev < -HEADING_DEADBAND &&
servoOutPrcnt[0 ] < 0))

  {

   servo_out[ SERVO_RIGHT_WINCH_3 ].value = servo_out[ SERVO_RIGHT_WINCH_3 ].value_failsafe

    + (fabs(servoOutPrcnt[0])*SERVO_R_WINCH_MAX_TRAVEL*SERVO_R_WINCH_SCALE_FACTOR);


   servo_out[ SERVO_LEFT_WINCH_4 ].value = servo_out[ SERVO_LEFT_WINCH_4 ].value_failsafe;

   //For DEBUG, disable next line

   Servos_Update_All();


   //flag set for left turn, but ONLY for a turn not caused by negative servo value

   if (headingDev > HEADING_DEADBAND && servoOutPrcnt[0] >= 0)

    {


        turnedRight[0] = 1;

        turnedLeft[0] = 0;

    }

   //DEBUG LINE BELOW
```

```
    //printf("\nSteering Right\n");
   }
  else
   {
    turnedLeft[0] = 0;
    turnedRight[0] = 0;

    //DEBUG LINE BELOW
  //printf("\nServos Unchanged -- In Heading Deadband\n");
   }

 //Sets steeringActive flag true if steering occurred earlier this loop and
 //disables the reset of steering scale to 1 until straight flight resumes
 if (headingDev > -HEADING_DEADBAND && headingDev < HEADING_DEADBAND)
   {
    steeringActive = 0;
   }
  else
   {
   steeringActive = 1;
   }

 //Counter for how many loops occur at minimum steer before steer output begins scaling
 //Intended to account for delay in heading change after initial steering input
 if (steeringActive)
   {
    steerDelayCounter += 1;
   }
```

```
 else
  {
    steerDelayCounter = 0;
  }



 //DEBUG LINE
 //printf("\n\nValues for this loop:\n steerDelayCounter (value for next loop): %d\n headingDev from actual-->desired:
%.3f\
//n achievedYawRate[0] (prev. loop to now): %.3f\n"
 //" desiredYawRate[0,1]: [%.3f, %.3f]\n steeringActive: %d\n",
 //steerDelayCounter, headingDev, achievedYawRate[0], desiredYawRate[0], desiredYawRate[1], steeringActive);


 //Writes current values to previous values for next loop
 desiredYawRate[1] = desiredYawRate[0];
 headingAct[1] = headingAct[0];
 headingTime[1] = headingTime[0];
 achievedYawRate[1] = achievedYawRate[0];


}


 //*****************************************************************//
 //Function: landingFlare
 //
 //Desc: Performs a simple flare maneuver when parafoil system altitude
 // is less than FLARE_HEIGHT above ground
 //
 //Receives:
```

```
 //
 //Returns:
 //
 //CONSTANTS: SERVO_RIGHT_WINCH_3, SERVO_LEFT_WINCH_4, SERVO_R_WINCH_MAX_TRAVEL,
 // SERVO_L_WINCH_MAX_TRAVEL, SERVO_R_WINCH_SCALE_FACTOR,
 // SERVO_L_WINCH_SCALE_FACTOR, FLARE_PRCNT
 //
 //Globals:
 //_____//



void landingFlare(void){
  //Sets both servos to a percentage of maximum pull to perform braking flare near landing
  servo_out[ SERVO_RIGHT_WINCH_3 ].value = servo_out[ SERVO_RIGHT_WINCH_3 ].value_failsafe
    + (FLARE_PRCNT*SERVO_R_WINCH_MAX_TRAVEL*SERVO_R_WINCH_SCALE_FACTOR);

  servo_out[ SERVO_LEFT_WINCH_4 ].value = servo_out[ SERVO_LEFT_WINCH_4 ].value_failsafe
    - (FLARE_PRCNT*SERVO_L_WINCH_MAX_TRAVEL*SERVO_L_WINCH_SCALE_FACTOR);

  //For DEBUG, disable next line
  Servos_Update_All();

  //DEBUG LINE
  //printf("\n\n***Initiate landing flare***\n Left Servo Value: %.3f\n Right Servo Value: %.3f\n", FLARE_PRCNT,
FLARE_PRCNT);
}

// Receives String latitude in format <degrees[(-90) to 90].min[2]sec[6]>; converts to decimal degrees
float convert_latitude_to_decimal_degrees(String latitude)
```

```
{
  float converted_latitude;



  String deg; //[0-2]
  String secs; //[3-]

  deg = latitude.substring(0,2);
  latitude.remove( 0, 2 );

  secs = latitude;

  converted_latitude = deg.toFloat() + (secs.toFloat() / 60.0 );

  //Serial.print("Converted: ");
  //Serial.println(converted_latitude);

  //Serial.print("Degrees: ");
  //Serial.println(deg);

  //Serial.print("Rest: ");
  //Serial.println(secs);

  return converted_latitude;
}



// Receives String longitude in format <degrees[(-180) to 180].min[2]sec[6]>; converts to decimal degrees
```

```
float convert_longitude_to_decimal_degrees(String longitude)
{
  float converted_longitude;

  String deg; //[0-2]
  String secs; //[3-]
  char neg = '-';

  if(longitude.charAt(0) == neg)
  {
    // Serial.println("IS NEGATIVE");
    longitude.remove(0,1);
  }

  deg = longitude.substring(0,3);
  longitude.remove( 0, 3 );

  secs = longitude;

  converted_longitude = deg.toFloat() + (secs.toFloat() / 60.0 );

  //Serial.print("Converted: ");
  //Serial.println(converted_longitude);

  //Serial.print("Degrees: ");
  //Serial.println(deg);

  //Serial.print("Rest: ");
```

```
  //Serial.println(secs);


  return -1 * converted_longitude;
}

void flight_control(String gpsData) // Written by Richard Park
{
  float gps_latitude;
  float gps_longitude;
  float gps_altitude;

  String string_lat;
  String string_long;
  String string_alt;

  static float prev_alt = 0;

  String s = gpsData;
  String delimiter = ",";
  String token;

  bool enable_control = false;

  int pos = 0;
  int count = 0;
  int comma_count = 0;
  static int j = 0; // counter
```

```
// Arbitrarily large array to store list;
String token_list[20];

// Parse, tokenize and store in array
while ( comma_count != 15 )
{
        comma_count++;
        pos = s.indexOf(",");
        token = s.substring(0, pos);
token_list[ count ] = token;

// 'Pops' item from front
s.remove( 0, pos + delimiter.length() );

count++;
}

// Grab the desired tokens
string_lat = token_list[2];
string_long = token_list[4];
string_alt = token_list[9];

// Sanity checks; if NULL; no update!
if (string_lat.length() != 0){
  gGPSlatitude = gps_latitude = convert_latitude_to_decimal_degrees(string_lat);
}

if (string_long.length() != 0){
```

```
  gGPSlongitude = gps_longitude = convert_longitude_to_decimal_degrees(string_long);
}

if (string_alt.length() != 0){
  gGPSaltitude = gps_altitude = string_alt.toFloat();
}

//gGPSlatitude = gps_latitude; // update global variables
//gGPSlongitude = gps_longitude;

Serial.println("Fixed lat: " + String(gGPSlatitude,DEC));
//gGPSaltitude = gps_altitude;
Serial.println(gps_latitude,5);
Serial.println(gps_longitude,5);
Serial.println(gps_altitude,5);
if((prev_alt > gGPSaltitude) && (millis() > 2000000)){ // 30000 seconds
  if(j > 3){
    simple_control();
    enable_control = true;
  }
  j++;
}
else j = 0;
if(enable_control){
  simple_control();
}

prev_alt = gps_altitude; // store previous altitude
```

```
}
//End Josh's control code


/* ************************************************************************
 * File Name:       MS5803_12BA.h
 * Author:          Stephen Wayne
 * Date:            February 11, 2015
 * Revised:

 * Header Description:
 ************************************************************************ */


#ifndef MS5803_12BA_H
 #define MS5803_12BA_H

 #include <Wire.h>
 #include "Arduino.h"
 #include <WProgram.h>
 #include <MS5803_I2C.h>

 #define R1_VAL 97900
 #define R2_VAL 198000

 void init_MS5803_12BA(void);
 float getTemp_F(word);
 float getTemp_F_LM19(word,char);
 double altitude(double, double);
 double sealevel(double, double);
```

```
  String poll_MS5803_12BA(void);


#endif

/* ******************************************************************************
 * File Name:       MS5803_12BA.cpp
 * Author:          Stephen Wayne
 * Created:         February 11, 2014
 * Revised:

 * Project Description: This is the driver file for the digital temp/pressure
 * sensor.
 ****************************************************************************** */
#include <MS5803_I2C.h>
//#include <Wire.h>
//#include <WProgram.h>
#include "MS5803_12BA.h"



MS5803 sensor(ADDRESS_HIGH);


//Create variables to store results
float temperature_c, temperature_f;
double pressure_abs, pressure_relative, altitude_delta, pressure_baseline;
double base_altitude = 1655.0; // Altitude of SparkFun's HQ in Boulder, CO. in (m)


void init_MS5803_12BA(void){
  //Retrieve calibration constants for conversion math.
```

```
  sensor.reset();
  sensor.begin();
  pressure_baseline = sensor.getPressure(ADC_4096);
}

String poll_MS5803_12BA(void)
{
  // Read temperature from the sensor in deg C. This operation takes about
  temperature_c = sensor.getTemperature(CELSIUS, ADC_512);
  // Read temperature from the sensor, convert to deg F
  temperature_f = sensor.getTemperature(FAHRENHEIT, ADC_512);
  // Read pressure from the sensor in mbar.
  pressure_abs = sensor.getPressure(ADC_4096);
  // Convert abs pressure with the help of altitude into relative pressure
  // This is used in Weather stations.
  pressure_relative = sealevel(pressure_abs, base_altitude);
  // Taking our baseline pressure at the beginning we can find an approximate
  // change in altitude based on the differences in pressure.
  altitude_delta = altitude(pressure_abs , pressure_baseline);

  Serial.print("Temperature F = ");
  Serial.println(temperature_f);

  Serial.print("Pressure abs (mbar)= ");
  Serial.println(pressure_abs);

  Serial.print("Pressure relative (mbar)= ");
  Serial.println(pressure_relative);
```

```
  Serial.print("Altitude change (m) = ");
  Serial.println(altitude_delta);

  return String(String(temperature_f, DEC) + "," + String(pressure_abs, DEC) + "," + String(pressure_relative,DEC) + "," +
String(altitude_delta,DEC));
}

double altitude(double P, double P0)
// Given a pressure measurement P (mbar) and the pressure at a baseline P0 (mbar),
// return altitude (meters) above baseline.
{
  return(44330.0*(1-pow(P/P0,1/5.255)));
}

double sealevel(double P, double A)
// Given a pressure P (mbar) taken at a specific altitude (meters),
// return the equivalent pressure (mbar) at sea level.
// This produces pressure readings that can be used for weather measurements.
{
  return(P/pow(1-(A/44330.0),5.255));
}

float getTemp_F(word analogTemp){
  float temp = 0;
  temp = analogTemp / 1023.0 * 1.2;
  temp = temp - 0.5;
  temp = temp / 0.01;
  temp = temp * 9/5 + 32;
```

```
  return temp;
}

float getTemp_F_LM19(word analogTemp, char charlie){
  float tempC = 0;
  float tempF = 0;
  if(charlie == 'R'){ // if remote data
    tempC = analogTemp*1.260/1023.0; //voltage measured at resistor divider. Compensate for extra volt drop
    tempC = tempC * (R1_VAL + R2_VAL) / R1_VAL;
  }
  else if(charlie == 'L'){ //if local data
    tempC = analogTemp*3.300/1023.0;
  }
  tempC = -1481.96+sqrt(2196200.0+(1.8639-tempC)/(0.00000388));
  tempF = tempC * 1.80 + 32.0;
  Serial.print("Degrees F: ");
  Serial.println(tempF);
  return tempF;
} /* ********************************************************************************
 * File Name:       Iridium.h
 * Original Author:  Gabe Pearhill
 * Modifying Author: Stephen Wayne
 * Date Created:     April 3, 2015
 * Revised:

 * Header Description:
 ********************************************************************************* */
```

```
#ifndef IRIDIUM_H
 #define IRIDIUM_H

 #include "Arduino.h"

 #define IRIDIUM_DATA_RATE 19200
 #define IRIDIUM_Serial Serial1

 // Function Prototypes
 void init_Iridium(void); //initialize Iridium
 String processIridium(String packet); // Iridium State machine
 String checkForIncomingPackets(void);
 int getIridiumResponse(void);
 // End of Iridium.h

#endif

/* ************************************************************************
 * File Name:       Iridium.cpp
 * Original Author:  Gabe Pearhill
 * Modifying Author: Stephen Wayne
 * Date Created:     April 3, 2015
 * Revised:

 * Project Description: This is the driver file for the Iridium 9603 SBD modem
 ************************************************************************ */
#include "Iridium.h"
#include "WCharacter.h"
```

```
const int iridiumPowerOn = 2;
const unsigned long IR_DELAY_TIME = 30000;

elapsedMillis iridiumTimer;

enum IridiumState {
    IR_DELAY,
    IR_ASKING_TO_SEND,
    IR_SENDING_DATA,
    IR_TRANSMITTING
};

enum IridiumResponse {
    IR_READY,
    IR_OK,
    IR_TRANS_RESP,
    IR_ERROR,
    IR_NULL,
    CUTDOWN,
    IR_ZERO
};

bool actionPerformed = false;

IridiumState iridiumState;

String iridiumResponse = "";
```

```
bool cutdown = false;


void init_Iridium(void){
  IRIDIUM_Serial.begin(IRIDIUM_DATA_RATE); //Open Iridium Serial at specified data rate
  pinMode(iridiumPowerOn, OUTPUT);
  digitalWrite(iridiumPowerOn, HIGH); // turn on iridium
  iridiumState = IR_DELAY; //initialize iridium state machine
}


String processIridium(String packet){
String recPacket = "";
  // Iridium State Machine
  switch(iridiumState){
    case IR_DELAY: //Allow iridium to boot for data-sheet specified time
       Serial.println("Iridium Booting/Delay");
       if(iridiumTimer > IR_DELAY_TIME){
         iridiumTimer = 0;
         iridiumState = IR_ASKING_TO_SEND;
         actionPerformed = false;
       }
       break;
    case IR_ASKING_TO_SEND: // tell iridium it will receive a packet to send
       Serial.println("ASK To Send State");
       if(!actionPerformed){
         IRIDIUM_Serial.println("AT+SBDWT"); // only send 120 bytes
         actionPerformed = true;
       }
```

```
    if(iridiumTimer > 1000 | getIridiumResponse() != IR_NULL){
      iridiumTimer = 0;
      iridiumState = IR_SENDING_DATA;
      actionPerformed = false;
    }
    break;
  case IR_SENDING_DATA: // Sending Iridium the packet
    Serial.println("Sending Data State");
    if(!actionPerformed){
      IRIDIUM_Serial.print(packet); // only print 120 bytes max
      actionPerformed = true;
    }
    if ((iridiumTimer > 1000) | (getIridiumResponse() != IR_NULL)){
      iridiumTimer = 0;
      iridiumState = IR_TRANSMITTING;
      actionPerformed = false;
    }
    break;
  case IR_TRANSMITTING: //Transmit the packet
    Serial.println("Transmitting State");
    if(!actionPerformed){
      IRIDIUM_Serial.println("AT+SBDI");
      actionPerformed = true;
    }
    if((iridiumTimer > 15000) | (getIridiumResponse() != IR_NULL)){
      iridiumTimer = 0;
      iridiumState = IR_DELAY;
      actionPerformed = false;
```

```
      }
      break;
  }
  if(iridiumState != IR_SENDING_DATA){
    recPacket = checkForIncomingPackets();
  }
  return recPacket;
}

String checkForIncomingPackets(){
String receivePacket = "";
  IRIDIUM_Serial.println("AT+SBDRB"); // ask for mobile terminated buffer data
  delay(10);
  char mt[64]; //read mobile terminated buffer
  int index = 0;
  for(int i = 0; i < 64;i++){
    mt[i] = '\0';
  }
  for (int i = 0; i < 64; i++){
    if(IRIDIUM_Serial.available()){
      mt[index] = IRIDIUM_Serial.read(); //get packet
      index++;
    }
    else{
      delay(1);
    }
  }
```

```
  if(mt[2] == 'c' && mt[3] == 'u' && mt[4] == 't' && mt[5] == 'd' && mt[6] == 'o'
       && mt[7] == 'w' && mt[8] == 'n'){
         cutdown = true;
  } //check for cutdown. first two bytes are checksum info. trigger cutdown
  if(mt[2] != '\0') receivePacket = String(mt);
  else receivePacket = "None";

  return receivePacket;
}

int getIridiumResponse(){
  if(iridiumState != IR_DELAY){ //if Iridium has booted
    for(int i = 0; i < 64; i++){
      if(IRIDIUM_Serial.available()){
        iridiumResponse += char(IRIDIUM_Serial.read()); //get incoming byte
      }
    }
    if(iridiumResponse.indexOf("READY") >= 0){
      Serial.print("Got READY\n");
      return IR_READY;
    }
    else if(iridiumResponse.indexOf("OK") >= 0){
      Serial.print("Got OK\n");
      return IR_OK;
    }
    else if(iridiumResponse.indexOf("SBD") >= 0){
      Serial.print("Got SBD\n");
      return IR_TRANS_RESP;
```

```
    }
    else if (iridiumResponse.indexOf("0") >= 0){
      Serial.print("Got 0\n");
      return IR_ZERO;
    }
    else if(iridiumResponse.indexOf("cutdown") >= 0){
      Serial.print("Got CUTDOWN\n");
      return CUTDOWN;
    }
  }
return IR_NULL;
}
```

```
/* ********************************************************************************
 * File Name:       IMU_9DOF.h
 * Author:          Stephen Wayne
 * Date:            February 11, 2015
 * Revised:

 * Header Description:
 ********************************************************************************* */

#ifndef IMU_9DOF_H
 #define IMU_9DOF_H

 #include <Wire.h>
 #include "Arduino.h"
 #include <WProgram.h>
 #include <MS5803_I2C.h>

 #define  ADXL345_ADDRESS (0xA6 >> 1)
 //There are 6 data registers, they are sequential starting
 //with the LSB of X.  We'll read all 6 in a burst and won't
 //address them individually
 #define ADXL345_REGISTER_XLSB (0x32)

 //Need to set power control bit to wake up the adxl345
 #define ADXL_REGISTER_PWRCTL (0x2D)
 #define ADXL_PWRCTL_MEASURE (1 << 3)
```

```
#define ITG3200_ADDRESS (0xD0 >> 1)
//request burst of 6 bytes from this address
#define ITG3200_REGISTER_XMSB (0x1D)
#define ITG3200_REGISTER_DLPF_FS (0x16)
#define ITG3200_FULLSCALE (0x03 << 3)
#define ITG3200_42HZ (0x03)

#define HMC5843_ADDRESS (0x3C >> 1)
//First data address of 6 is XMSB.  Also need to set a configuration register for
//continuous measurement
#define HMC5843_REGISTER_XMSB (0x03)
#define HMC5843_REGISTER_MEASMODE (0x02)
#define HMC5843_MEASMODE_CONT (0x00)

void init_IMU_9DOF(void);
String read_IMU_9DOF(void);
void i2c_write(int,byte,byte);
void i2c_read(int,byte,int,byte*);
void init_adxl345(void);
void read_adxl345(void);
void init_itg3200(void);
void read_itg3200(void);
void init_hmc5843(void);
void read_hmc5843(void);

#endif
```

```
/* ************************************************************************
 * File Name:       IMU_9DOF.cpp
 * Author:          Stephen Wayne
 * Created:         February 11, 2015
 * Revised:

 * Project Description: This is the driver file for the 9DOF IMU from Sparkfun
 ************************************************************************ */
//#include <Wire.h>
//#include <WProgram.h>
#include "IMU_9DOF.h"



int accel_data[3];
int gyro_data[3];
int mag_data[3];



void init_IMU_9DOF(){
  for(int i = 0; i < 3; ++i) {
    accel_data[i] = mag_data[i] = gyro_data[i] = 5;
  }
  init_adxl345();
  init_hmc5843();
  init_itg3200();
  Serial.print("IMU Initialized");
```

```
}

String read_IMU_9DOF(){
  read_adxl345();
  Serial.print("ACCEL: ");
  Serial.print(accel_data[0]);
  Serial.print(",");
  Serial.print(accel_data[1]);
  Serial.print(",");
  Serial.println(accel_data[2]);

  read_hmc5843();
  Serial.print("MAG: ");
  Serial.print(mag_data[0]);
  Serial.print(",");
  Serial.print(mag_data[1]);
  Serial.print(",");
  Serial.println(mag_data[2]);

  read_itg3200();
  Serial.print("GYRO: ");
  Serial.print(gyro_data[0]);
  Serial.print(",");
  Serial.print(gyro_data[1]);
  Serial.print(",");
  Serial.println(gyro_data[2]);
  Serial.println();
  //Sample at 3.33Hz*/
```

```
  return
String(String(accel_data[0],DEC)+","+String(accel_data[1],DEC)+","+String(accel_data[2],DEC)+","+String(mag_data[0],DEC)+","+String(mag_data[1],DEC)+","+String(mag_data[2],DEC)+","+String(gyro_data[0],DEC)+","+String(gyro_data[1],DEC)+","+String(gyro_data[2],DEC));
}

void i2c_write(int address, byte reg, byte data){
  // Send output register address
  Wire.beginTransmission(address);
  Wire.write(reg);
  // Connect to device and send byte
  Wire.write(data); // low byte
  Wire.endTransmission();
}

void i2c_read(int address, byte reg, int count, byte* data){
  int i = 0;

  // Send input register address
  Wire.beginTransmission(address);
  Wire.write(reg);
  Wire.endTransmission();
  // Connect to device and request bytes

  Wire.beginTransmission(address);
  Wire.requestFrom(address,count);
  while(Wire.available()){ // slave may send less than requested
    char c = Wire.read(); // receive a byte as character
    data[i] = c;
```

```
    i++;
  }
  Wire.endTransmission();
}

void init_adxl345(){
  byte data = 0;

  i2c_write(ADXL345_ADDRESS, ADXL_REGISTER_PWRCTL, ADXL_PWRCTL_MEASURE);

  //Check to see if it worked!
  i2c_read(ADXL345_ADDRESS, ADXL_REGISTER_PWRCTL, 1, &data);
  Serial.println((unsigned int)data);
}

void read_adxl345(){
  byte bytes[6];
  memset(bytes,0,6);

  //read 6 bytes from the ADXL345
  i2c_read(ADXL345_ADDRESS, ADXL345_REGISTER_XLSB, 6, bytes);

  //now unpack the bytes
  for (int i=0;i<3;++i) {
    accel_data[i] = (int16_t)((int)bytes[2*i] + (((int)bytes[2*i + 1]) << 8));
  }
}
```

```
void init_itg3200(){
  byte data = 0;

  //Set DLPF to 42 Hz (change it if you want) and
  //set the scale to "Full Scale"
  i2c_write(ITG3200_ADDRESS, ITG3200_REGISTER_DLPF_FS, ITG3200_FULLSCALE | ITG3200_42HZ);

  //Sanity check! Make sure the register value is correct.
  i2c_read(ITG3200_ADDRESS, ITG3200_REGISTER_DLPF_FS, 1, &data);

  Serial.println((unsigned int)data);
}

void read_itg3200(){
  byte bytes[6];
  memset(bytes,0,6);

  //read 6 bytes from the ITG3200
  i2c_read(ITG3200_ADDRESS, ITG3200_REGISTER_XMSB, 6, bytes);  //now unpack the bytes
  for (int i=0;i<3;++i) {
  gyro_data[i] = (int16_t)((int)bytes[2*i + 1] + (((int)bytes[2*i]) << 8));
  }
}

void init_hmc5843(){
  byte data = 0;
  //set up continuous measurement
  i2c_write(HMC5843_ADDRESS, HMC5843_REGISTER_MEASMODE, HMC5843_MEASMODE_CONT);
```

```
  //Sanity check, make sure the register value is correct.
  i2c_read(HMC5843_ADDRESS, HMC5843_REGISTER_MEASMODE, 1, &data);
  Serial.println((unsigned int)data);
}

void read_hmc5843(){
  byte bytes[6];
  memset(bytes,0,6);

  //read 6 bytes from the HMC5843
  i2c_read(HMC5843_ADDRESS, HMC5843_REGISTER_XMSB, 6, bytes);

  //now unpack the bytes
  for (int i=0;i<3;++i) {
  mag_data[i] = (int16_t)((int)bytes[2*i + 1] + (((int)bytes[2*i]) << 8));
  }
}
```

```
/* ************************************************************************
 * File Name:       GPS.h
 * Author:          Stephen Wayne
 * Date Created:    April 1, 2015
 * Revised:
 ************************************************************************ */

#ifndef GPS_H
 #define GPS_H

 #include "Arduino.h"

 #define GPS_DATA_RATE 9600
 #define GPS_Serial Serial3

 // Function Prototypes
 void init_GPS(void); //initialize GPS
 bool poll_GPS(String *string); //read data from GPS
 // End of GPS.cpp

#endif
```

```
/* ********************************************************************************
 * File Name:       GPS.cpp
 * Author:          Stephen Wayne
 * Created:         February 11, 2015
 * Revised:

 * Project Description: This is the driver file for the xxx GPS module
 ********************************************************************************* */
#include "GPS.h"
#include "WCharacter.h"

String GPS_Format = "$GPGGA"; // main GPS data
String Vel_Format = "$GPRMC"; // pull speed over ground data from this

void init_GPS(void){
  GPS_Serial.begin(GPS_DATA_RATE); //Open GPS Serial at specified data rate
}

bool poll_GPS(String *last_GPS) // main GPS function
{
static String curr_GPS, VelStr;
static bool newGPS_data = false, newVel_data = false;
const char s[2] = ","; // used to parse string
char *token;
```

```
char c = GPS_Serial.read(); //read in GPS output char by char
if(isAlphaNumeric(c)||(c=='\n')||(c==',')||(c=='$')||(c=='.')){ // if character is valid
  curr_GPS += c; //append to GPS buffer
}
if(c=='\n'){ //if the end of a line
  if(!strncmp(curr_GPS.c_str(),Vel_Format.c_str(),6)){ //pull velocity data
    newVel_data = true;
    token = strtok(const_cast<char*>(curr_GPS.c_str()),s);
    for(int i = 0; i < 7; i++){ // grab 8th CSV, which is speed over land
      token = ""; // clear the value initially
      token = strtok(NULL,s);
    }
    VelStr = token;
  }
  if(!strncmp(curr_GPS.c_str(),GPS_Format.c_str(),6)){ //if the correct sentence type
    newGPS_data = true; // indicate new data to store
    curr_GPS = strtok(const_cast<char*>(curr_GPS.c_str()),"\n"); // Trim newline from string
    if(newVel_data){
      *last_GPS = curr_GPS + s + VelStr + "\n"; //send GPGGA string back to main with velocity
      newVel_data = false;
    }
    else{
      *last_GPS = curr_GPS + ",NoNewVel" + "\n"; //send GPGGA string back to main w/o velocity
    }
    curr_GPS = ""; // reset GPS buffer
  }
  else{//Throw out anything else
    newGPS_data = false; //no new data to store
```

```
      curr_GPS = ""; //reset GPS buffer
    }
  }
  return newGPS_data; //return new data status
}
```

```
#include "XbeeIO.h"
#include <SD.h>
#include <SPI.h>
#include <Wire.h>
#include <Servo.h>
#include <math.h>
#include <MS5803_I2C.h>
#include "MS5803_12BA.h" // Driver libraries for various components
#include "IMU_9DOF.h"
#include "GPS.h"
#include "Iridium.h"
#include "simple_control.h"

boolean sdWrite(File,String,String);
boolean delayMs(long int, int);
void init_SD(void);

const int chipSelect = 15; //microSD chip select line
const int analogTemp = 22; // analog temperature input
const int analogPress = 23; // analog pressure input
File xbeeFile, logFile, gpsFile, iridFile; // files to be written to SD card
String xbeeFile_name = "xbeefile",logFile_name = "logfile", gpsFile_name = "gpsfile"; //8 characters or fewer
String iridFile_name = "iridfile"; // file names for SD card

int i;
```

```
void setup(){
  Wire.begin(); //initialize I2C communication
  Serial.begin(115200); // initialize debugging (USB) serial
  init_Iridium(); // initialize Iridium 9603 modem
  init_XBee(); // initialize Coordinator API xbee
  init_MS5803_12BA(); //Initialize Digital Temp/Pressure Sensor
  init_IMU_9DOF(); //Initialize Accelerometer, Gyroscope, Magnetometer
  init_GPS(); //Initialize GPS
  init_SD(); //Initialize microSD card
  init_Servos(); // initialize servos, reset position
  delay(1000); //wait for everything to initialize
}

void loop(){
static String xbee_print, datalog_print, dl_print_tmp, imu, GPS_data;
String iridPacket = "", receivePacket = "";
static int numBytes;
static byte XBee_buf[30];
static long int last_poll_pres = 0;
static long int last_poll_imu = 0;
static long int last_poll_gps = 0;
static long int last_poll_iridium = 0;
static int poll_time_pres = 2000; //milliseconds between polling Pressure and Temp
static int poll_time_imu = 1000; //milliseconds between polling IMU
static int poll_time_gps = 2000; //milliseconds between polling GPS
static int poll_time_iridium = 15000;
bool newGPS_data = false;
word pressVal = 0;
```

```
word tempVal = 0;
float tempF = 0;
float tempPress = 0;

float lat;
float longt;

 newGPS_data = poll_GPS(&GPS_data); // check if new data and update string
 if(newGPS_data && delayMs(last_poll_gps,poll_time_gps)){
  last_poll_gps = millis();
  Serial.println("New GPS Data:"); // debugging (next line also)
  Serial.println(GPS_data);
  sdWrite(gpsFile,gpsFile_name,GPS_data); // write updated GPS data
  flight_control(GPS_data); // update winches for navigation
 }

 if(delayMs(last_poll_pres, poll_time_pres)){ // Poll local sensors
  last_poll_pres = millis();
  datalog_print = poll_MS5803_12BA();
  datalog_print = String(String(last_poll_pres,DEC) + "," + datalog_print);
  dl_print_tmp = datalog_print;
  imu = read_IMU_9DOF();
  datalog_print = datalog_print + "," + imu;
  pressVal = analogRead(analogPress);
  tempPress = get_pressure(pressVal,'L'); // local
  tempVal = analogRead(analogTemp);
  tempF = getTemp_F_LM19(tempVal,'L'); // local
  datalog_print = datalog_print + "," + String(tempF,DEC) + "," + String(tempPress,DEC);
```

```
  sdWrite(logFile, logFile_name, datalog_print);
}

if(XBee_Serial.available()>20){ // if complete XBee receive data frame
  numBytes = XBee_Serial.available(); // length of data frame
  for(int i = 0; i < numBytes; i++){
    XBee_buf[i] = XBee_Serial.read(); // loop and store xbee Rx data in buffer
    if(XBee_buf[i] == -1){ // check if over-run
      Serial.println("No more bytes to read");
      Serial.println(numBytes);
      for(int j = 0; j < numBytes; j++){
        Serial.print(XBee_buf[j]);
      }
    }
  }
  Serial.println();
  xbee_print = read_Xbee(XBee_buf); // pull relevant data
  sdWrite(xbeeFile, xbeeFile_name, xbee_print);
  Serial.println();
}

if(delayMs(last_poll_iridium, poll_time_iridium)){
  last_poll_iridium = millis();
  iridPacket = String(GPS_data + "," + dl_print_tmp); // join strings, CSV
  receivePacket = processIridium(iridPacket); // send/receive data
  if(strcmp(receivePacket.c_str(),"None")){ // if there was a received string
    Serial.println("String received from Iridium"); // debug
    receivePacket = String(last_poll_iridium,DEC) + "," + "Rx" + "," + receivePacket;
```

```
    sdWrite(iridFile,iridFile_name,receivePacket); // store received data
    // could add functionality to send commands to capsule here
   } // only do this if it actually sends through iridium
   iridPacket = String(last_poll_iridium,DEC) + "," + "Tx" + "," + iridPacket;
   sdWrite(iridFile,iridFile_name,iridPacket); // store transmitted data
  }
}

void init_SD(void){ // initialize SD card
  pinMode(chipSelect, OUTPUT); // SD card on SPI
  if(!SD.begin(chipSelect)){ // if initialization error
   Serial.println("initialization failed!");
   return;
  }
  else{
   Serial.println("SD initialization successful");
   sdWrite(xbeeFile, xbeeFile_name, String("time(ms),xbee_addr,temp(f),press(psi)"));
   sdWrite(logFile, logFile_name,
String("time_ms,dTemp(f),pres_abs(mbar),press_rel(mbar),alt_change(m),accelx,accely,accelz,magx,magy,magz,gyrox,g
yroy,gyroz,aTemp(f),aPres(psi)"));
   sdWrite(gpsFile, gpsFile_name,
String("identifier,time,lat,dir,long,dir,fix,numSat,precision,alt,meter,height,meter,blank,check,vel(knots)"));
  }
}

// Function to easily write data to SD card
boolean sdWrite(File writeFile, String fileName, String write_string){
  String temp = String(fileName) + ".txt";
  char filename[temp.length()+1];
```

```
  temp.toCharArray(filename, sizeof(filename));
  writeFile = SD.open(filename, FILE_WRITE);
  writeFile.println(write_string);
  writeFile.close();
}

boolean delayMs(long int last_poll, int poll_time){ // Used for non-blocking delays
  if(millis() >= (last_poll + poll_time)) return true;
  else return false;
}
```

Appendix B. GUI Code

```
##### run.py #########
#!/usr/bin/env python3
import os
from application import app
app.run(host=os.getenv('IP', '0.0.0.0'),port=int(os.getenv('PORT', 8080)),debug=True)
################# imap_gmail_script.py ##################
#!/usr/bin/python3
import getpass
import imaplib
import email
import os
from datetime import datetime
import pytz
import socket


def format_time(time):
    if time:
        hours = str((int(time[0:2])-7)%24)
        if len(hours) < 2:
            hours = '0'+hours
        mins = time[2:4]
        if len(mins) < 2:
            hours = '0'+hours
        secs = time[4:6]
        if len(secs) < 2:
            hours = '0'+hours
        time = hours + ':' + mins + ':' + secs
    return time

def format_longitude(longitude, card):
    if longitude:
        deg = int(longitude[0:3])
        sec = float(longitude[3:])
        longitude =  deg+sec/60
        if card is 'W':
            longitude = longitude * -1
            return str(longitude)[:10]
        else:
            return str(longitude)[:9]


def format_latitude(latitude, card):
    if latitude:
        deg = int(latitude[0:2])
        sec = float(latitude[2:])
        latitude = deg+sec/60
        if card is 'S':
            latitude = latitude*-1
            return str(latitude)[:9]
        else:
            return str(latitude)[:8]


def attachment_handler(attachment):
    diag_dict = {}
    extra_data = attachment.split('/')
    gps_sentence = extra_data[0].split(',')
    diag_dict['Pressure'] = extra_data[2]
```

```
    diag_dict['Temperature'] = extra_data[1]
    if 'not' in extra_data[3]:
        diag_dict["cutdown"] = extra_data[3]
    else:
        diag_dict["cutdown"] = extra_data[3]
    for num, entry in enumerate(gps_sentence):
        if entry != '$GPGGA':# This approach works as long as we use this format
            continue
        else:
            diag_dict['Time'] = format_time(gps_sentence[num+1])
            diag_dict['Latitude'] = format_latitude(gps_sentence[num+2],
                                                    gps_sentence[num+3])
            diag_dict['Longitude'] = format_longitude(gps_sentence[num+4],
                                                      gps_sentence[num+5])
            diag_dict['Satellites'] = gps_sentence[num+7]
            diag_dict['Altitude'] = gps_sentence[num+9]

    return diag_dict

def search_mail(mail_handle):
    ############ Create a query to search for new mail ############
    form = "%d-%b-%Y"
    date = datetime.now(tz=pytz.utc)
    date = date.astimezone(pytz.timezone('US/Pacific'))
    s = date.strftime(form)
    query = "(FROM \"uivast1\" SINCE \"" + s + "\")"
    # print(query)
    typ, data = mail_handle.search(None, query)

    if typ != 'OK':                                  #typ is for error checking
       print("Error searching mail.")                #data is a tuple of msg num

    return data

def fetch_mail(mail_handle, msgId):
    typ, messageParts = mail_handle.fetch(msgId, '(RFC822)')
    if typ != 'OK':
        print("Error fetching mail.")
    emailBody = messageParts[0][1]
    mail = email.message_from_bytes(emailBody)
    diagnostics = []
    for part in mail.walk():
        if part.is_multipart():
            continue
        if part.get('Content-Disposition') is None:
            continue
        if part.get('Content-Disposition') == "inline":
            continue
        attachment = part.get_payload(decode=True).decode("utf-8", "ignore")
        diagnostics = attachment_handler(attachment)
    return diagnostics


def main():
    ############ Open IMAP connection to Gmail ##############
    M = imaplib.IMAP4_SSL("imap.gmail.com")
    M.login("Guidedparafoilsystem@gmail.com", "SeniorDesign2015")
    M.select()
    msgID = 0
    data = search_mail(M)
```

```
    if not data[0]:
        return {}
    msgID = data[0].split()[-1]
    diag = fetch_mail(M, msgID)
    diag['msg'] = str(int(msgID))
    M.close()
    M.logout()
    return diag
################ views.py ###############
from application import app
from flask import render_template
from flask import jsonify
from application.imap_gmail_script import main

@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html')

@app.route('/update')
def update():
    return jsonify(main())
<!-- index.html -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="static/css/main.css">
        <script
src="https://maps.googleapis.com/maps/api/js?v=3&key=AIzaSyAlKWprO3ovys6MzjsbMABBPZbQ
5i6C_lg"></script>

        <title>Guided Parafoil System Dashboard</title>
    </head>
    <body>

        <header>
            <img id = "nasalogo" src="static/nasa-logo.svg" alt="">
            <h1 id = "title">GPS Dashboard</h1>
            <img id="vandallogo" src="static/vandal_logo.jpg" alt="">
        </header>
        <div id="main-container">
            <input type="hidden" name="msgnum" id="msgnum" value="-1">
            {% if time %}
            <h3>Time Last Received: <span id="time">{{ time }} </span> PST</h3>
            {% else %}
            <h3>Time Last Received: <span id="time">N/A </span></h3>
            {% endif %}
            <!-- End of Time -->
            <h4>Cutdown Status: <span id="cd-stat">Not Cutdown</span></h4>
            {% if sats %}
            <h4>Number of Satellites: <span id="sats-val">{{ sats }}</span></h4>
            {% else %}
            <h4>Number of Satellites: <span id="sats-val">{{ sats }}</span></h4>
            {% endif %}
            <!-- End of Satellites -->
```

```
            <div id="gauges">
                <div id="altitude">
                    {% if alt %}
                    <h4>Altitude: <span id="altitude-val">{{ alt }}</span> m</h4>
                    {% else %}
                    <h4>Altitude: <span id="altitude-val">{{ alt }}</span> m</h4>
                    {% endif %}
                </div>
                <div id="temperature">
                    {% if temp %}
                    <h4>Temperature: <span id="temp-val">{{ temp }} </span>&deg;
C</h4>
                    {% else %}
                    <h4>Temperature: <span id="temp-val">{{ temp }} </span>&deg;
C</h4>
                    {% endif %}
                </div>
                <div id="pressure">
                    {% if temp %}
                    <h4>Pressure: <span id="pres-val">{{ pressure }} </span>
mbars</h4>
                    {% else %}
                    <h4>Pressure: <span id="pres-val">{{ pressure }} </span>
mbars</h4>
                    {% endif %}
                </div>
            </div>

            {% if latitude and longitude %}
            <h3>GPS Coordinates: <span id="lat-val">{{ latitude }}</span>, <span
id="long-val">{{ longitude }}</span></h3>
            {% else %}
            <h3>GPS Coordinates: <span id="lat-val">{{ latitude }}</span>, <span
id="long-val">{{ longitude }}</span></h3>
            {% endif %}

            <div id="map-canvas">
            </div>

        </div> <!--main-container-->

        <script src="static/js/vendor/jquery-1.11.1.min.js"></script>
        <script src="https://wzrd.in/standalone/d3-gauge"></script>
        <script src="https://wzrd.in/standalone/xtend"></script>
        <script src="static/js/d3gauge.js"></script>
        <script src="static/js/main.js"></script>
    </body>
</html>
html, body {
  padding: 0;
  margin: 0;
}

#main-container h4, h3 {
  text-align: center;
  margin-bottom: 0;
}

#main-container {
  margin: 0 5%;
```

```
}

#nasalogo, #vandallogo {
  height: 110px;
  width: auto;
}

#gauges {
  display: flex;
  justify-content: space-around;
  align-items: center;
}

#map-canvas {
  height: 350px;
}

header {
  border-bottom: 2px solid #000;
  margin: 0 auto 10px auto;
  display: flex;
  justify-content: space-around;
  align-items: center;
}

@media all and (max-width: 500px){
  #gauges {
    flex-direction: column;
  }

}
.d3-gauge {
  margin: 0 auto;
}

/* Simple */
.d3-gauge.simple .outer-circle {
  fill         :  #ccc;
  stroke       :  #000;
  stroke-width :  0.5px;
}

.d3-gauge.simple .inner-circle {
  fill         :  #fff;
  stroke       :  #E0E0E0;
  stroke-width :  2px;
}

.d3-gauge.simple .label {
  fill      :  #333;
  font-size :  18px;
}

.d3-gauge.simple .major-tick {
  stroke       :  #333;
  stroke-width :  2px;
}

.d3-gauge.simple .minor-tick {
  stroke       :  #666;
```

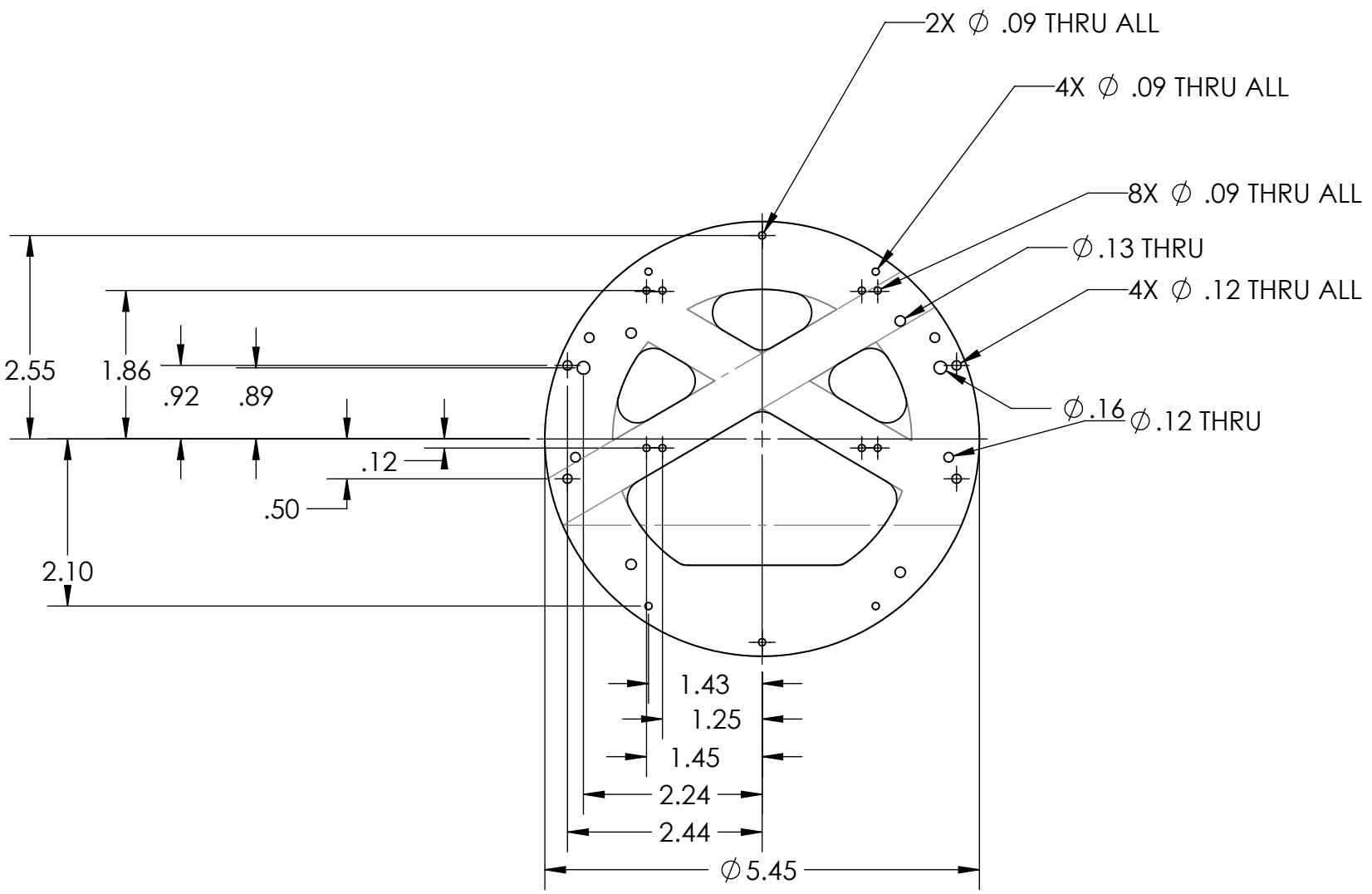
```
  stroke-width :  1px;
}

.d3-gauge.simple .major-tick-label {
  fill         :  darkblue;
  stroke-width :  2px;
  font-size    :  15px;
}

.d3-gauge.simple .needle {
  fill         :  #dc3912;
  stroke       :  #c63310;
  fill-opacity :  0.7;
}

.d3-gauge.simple .needle-container {
  fill         :  #4684EE;
  stroke       :  #666;
  fill-opacity :  1;
}

.d3-gauge.simple .current-value {
  fill         :  #000;
  stroke-width :  0px;
}

.d3-gauge.simple .green-zone {
  fill :  #FF9900
}

.d3-gauge.simple .yellow-zone {
  fill :  #FF9900
}

.d3-gauge.simple .red-zone {
  fill :  #DC3912
}

.d3-gauge.simple .blue-zone {
  fill: #3E97E0
}

.d3-gauge.simple .lightblue-zone {
  fill: #00CBFF
}
////////////// d3gauge.js //////////////////
// d3Gauge and xtend get pulled in via browserify-cdn standalone
// see script tags
var gauges = [];
var small = {
  size: 100,
  min: 0,
  max: 50,
  transitionDuration: 500

  ,
  label: 'label.text',
  minorTicks: 4,
  majorTicks: 5,
  needleWidthRatio: 0.6,
```

```
  needleContainerRadiusRatio: 0.7

  ,
  zones: [{
    clazz: 'yellow-zone',
    from: 0.73,
    to: 0.9
  }, {
    clazz: 'red-zone',
    from: 0.9,
    to: 1.0
  }]
};

function createGauge(opts, target) {
  var el = document.createElement('div');
  el.setAttribute('class', 'gauge-container');
  target.append(el);
  var g = d3Gauge(el, opts);
  g.currentValue = g._range / 2;
  gauges.push(g);
}

function getRandomNextValue(gauge) {
  gauge.currentValue += (Math.random() - 0.5) * gauge._range / 10;
  if (gauge.currentValue < gauge._min) gauge.currentValue = gauge._min;
  if (gauge.currentValue > gauge._max) gauge.currentValue = gauge._max;
  return gauge.currentValue;
}

function updateGauges() {
  gauges.forEach(function(gauge) {
    gauge.write(getRandomNextValue(gauge));
  });
}

createGauge({
  clazz: 'simple',
  label: '',
  max: 20000,
  size: 220,
  zones: [{
    clazz: 'yellow-zone',
    from: 0,
    to: 0
  }, {
    clazz: 'red-zone',
    from: 0,
    to: 0
  }]
}, $('#altitude'));

createGauge({
  clazz: 'simple',
  label: '',
  min: -60,
  max: 30,
  size: 220,
  zones: [{
    clazz: 'blue-zone',
```

```
    from: 0,
    to: .15
  }, {
    clazz: 'lightblue-zone',
    from: .15,
    to: .25
  }]
}, $('#temperature'));

createGauge({
  clazz: 'simple',
  label: '',
  max: 1050,
  size: 220,
  zones: [{
    clazz: 'yellow-zone',
    from: 0,
    to: 0
  }, {
    clazz: 'red-zone',
    from: 0,
    to: 0
  }]
}, $('#pressure'));
///////// main.js ///////////
var poly;
var map;
var msg;

function get_data() {
  $.ajax({
    dataType: "json",
    url: "/update",
    success: function(data){
        msg = $("#msgnum").val();
        if(data["msg"] == msg){
          console.log("No new data");
        }else{
          console.log(data);
          console.log(data["msg"], msg)
          $("#msgnum").val(data["msg"]);
          if(data["Altitude"]){
            $("#altitude-val").text(data['Altitude']);
            gauges[0].write(data['Altitude']);
          }
          if(data["Satellites"]){
            $("#sats-val").text(data['Satellites']);
          }
          if(data["Time"]){
            $("#time").text(data['Time']);
          }
          if(data["Latitude"] && data["Longitude"]){
            $("#lat-val").text(data['Latitude']);
            $("#long-val").text(data['Longitude']);
            addLatLng(data['Latitude'], data['Longitude']);
          }
          if(data["Pressure"]){
            $("#pres-val").text(data["Pressure"]);
            gauges[2].write(data["Pressure"]);
          }
```

```
          if(data["Temperature"]){
            $("#temp-val").text(data["Temperature"]);
            gauges[1].write(data["Temperature"]);
          }
          if(data["cutdown"] == "cutdown"){
            $("#cd-stat").text("Cutdown!");
          }
        }
    }
  });
}

function initialize() {
  var mapOptions = {
    zoom:  15,
    // Center the Map over the Northwest
    center: new google.maps.LatLng(46.75320, -118.30832),
    mapTypeId:"hybrid"
  };

  map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);

  var polyOptions = {
    strokeColor: '#000000',
    strokeOpacity: 1.0,
    strokeWeight: 3
  };
  poly = new google.maps.Polyline(polyOptions)
  poly.setMap(map);
}

function addLatLng(Lat, Lng) {
  var path = poly.getPath();

  coords = new google.maps.LatLng(Lat, Lng);
  path.push(coords);
  map.panTo(coords);
  var marker = new google.maps.Marker({
    position: coords,
    title: '#' + path.getLength(),
    map: map
  });
}

google.maps.event.addDomListenerOnce(window, 'load', initialize);
setInterval(get_data, 4000);
console.log(gauges);
```

Appendix C. Engineering Drawing



DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum

GPS

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

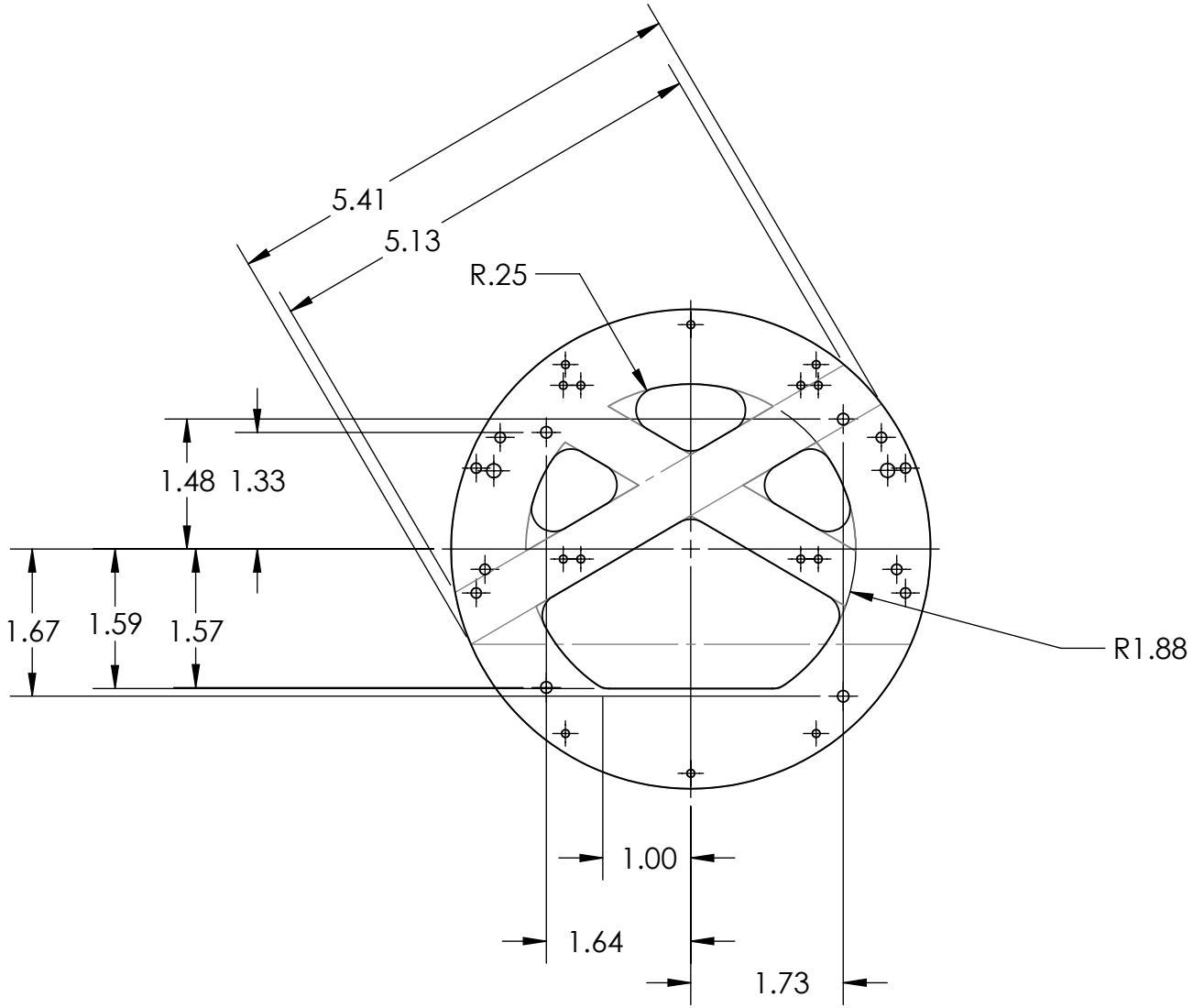
DRAWN BY: Brian Kisling DATE: 5/10/2015

FILE NAME: control system chassis top.SLDPRT

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 1:2 SHEET: 1 OF 2



DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum

GPS

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

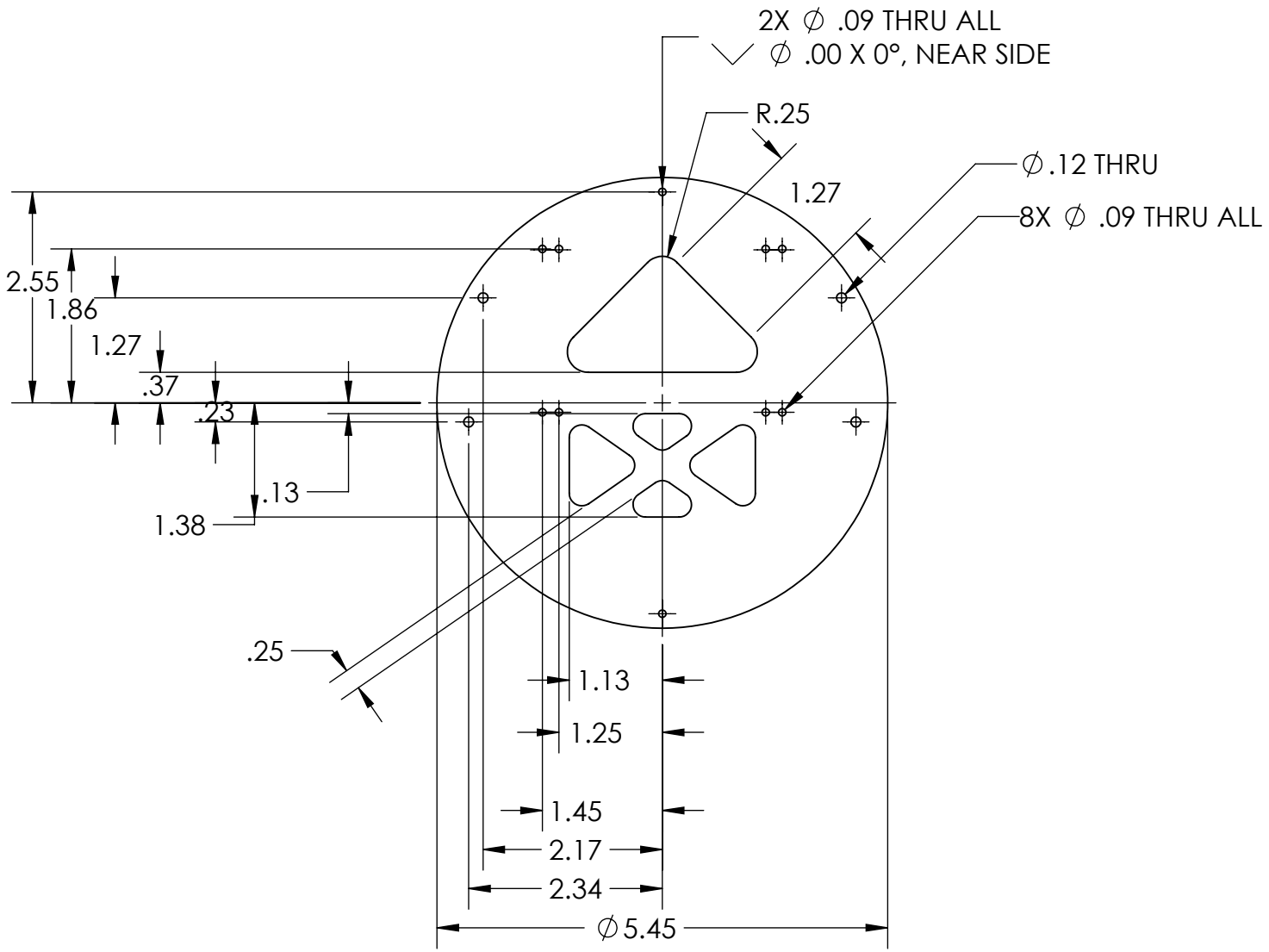
DRAWN BY: Brian Kisling DATE: 5/10/2015

FILE NAME: control system chassis top.SLDPRT

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 1:2 SHEET: 2 OF 2



DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum

GPS

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

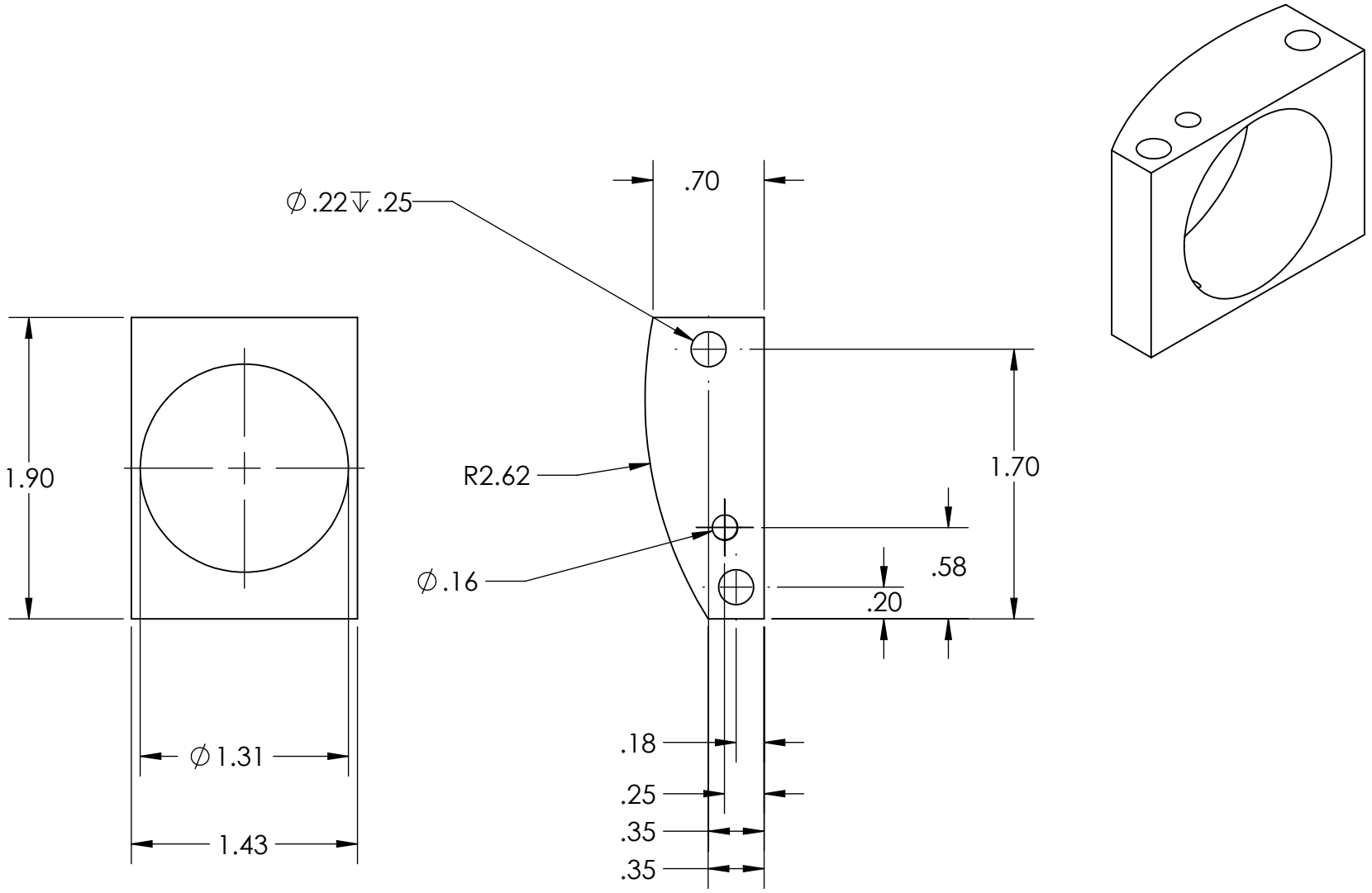
DRAWN BY: Brian Kisling DATE: 5/10/2015

FILE NAME: control system chassis bottom.SLDPRT

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 1:2 SHEET: 1 OF 1



PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UNIVERSITY OF IDAHO, ME DEPARTMENT. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UNIVERSITY OF IDAHO, ME DEPARTMENT IS PROHIBITED.

DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum

GPS

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

DRAWN BY: Brian Kisling DATE: 4/29/2015

FILE NAME: 01-06 Pulley Housing right-V2.SLDPRT

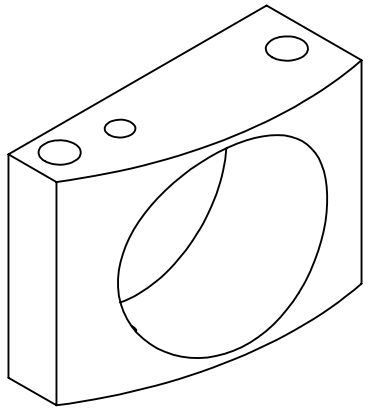
UNIVERSITY OF IDAHO
ME DEPARTMENT

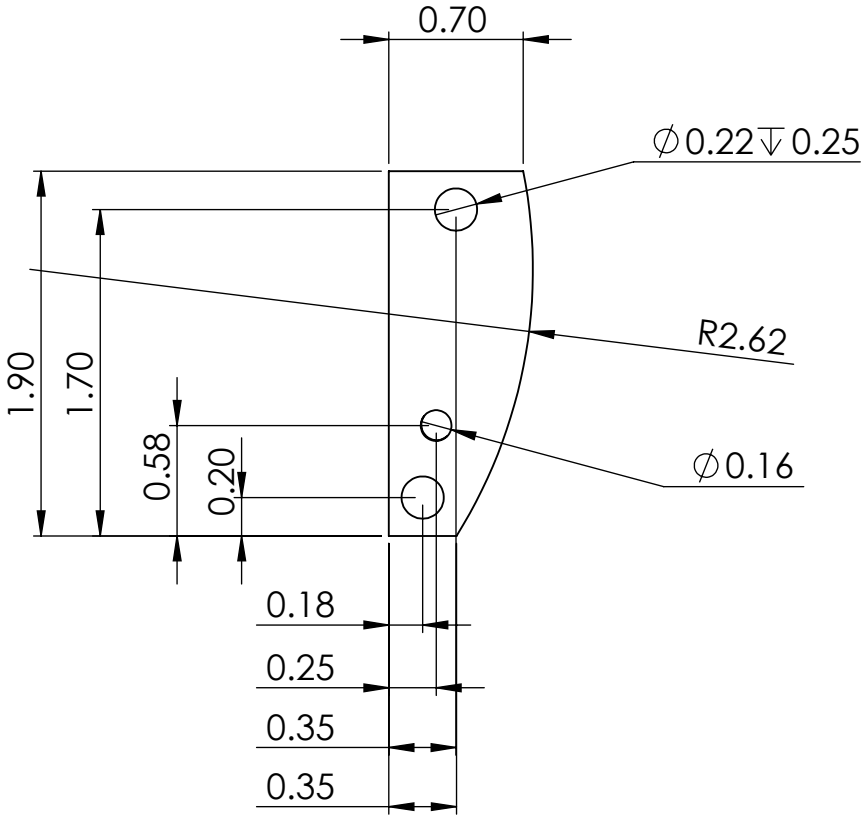
PART #:

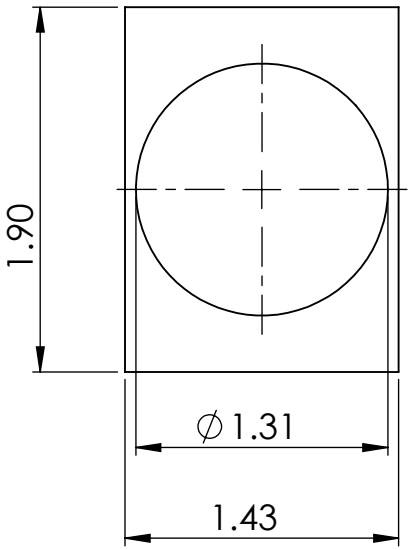
QTY:

SCALE: 1:1

SHEET: 1 OF 1







DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: 3-D printed plastic

GPS

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

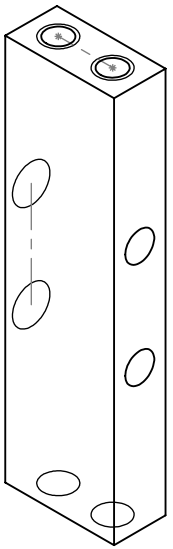
DRAWN BY: Brian Kisling DATE: 4/29/2015

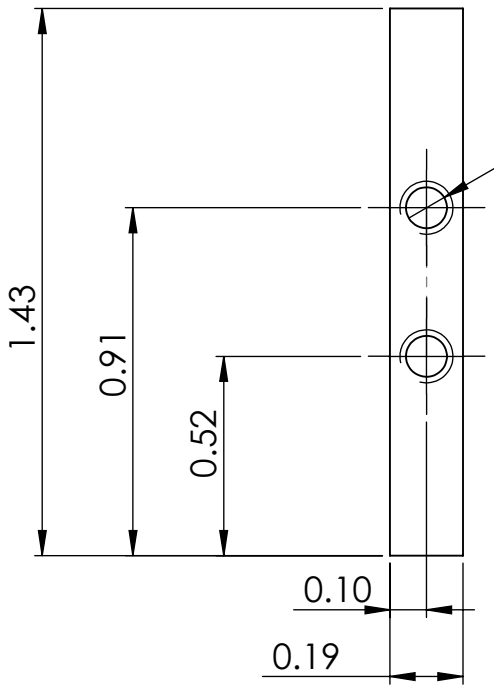
FILE NAME: 01-06 Pulley Housing Left-V2.SLDPRT

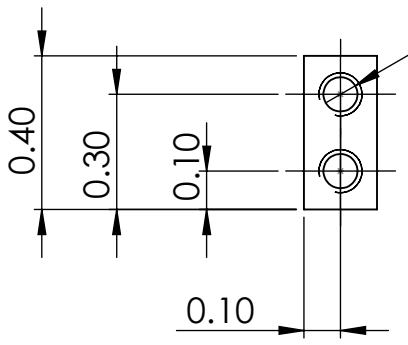
UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 1:1 SHEET: 1 OF 1







DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum

GPS

DEFAULT TOLERANCES:

LINEAR: X. ± .25, X.X ± .1, X.XX ± .01, X.XXX ± .002

ANGULAR: X. ± 2, X.X ± 1, X.XX ± 0 30'

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

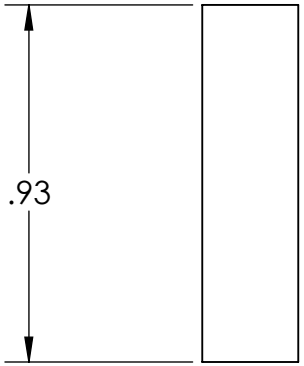
DRAWN BY: Brian Kisling DATE: 5/10/2015

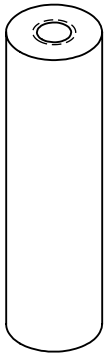
FILE NAME: 01-05 Support Strut-V2.SLDPRT

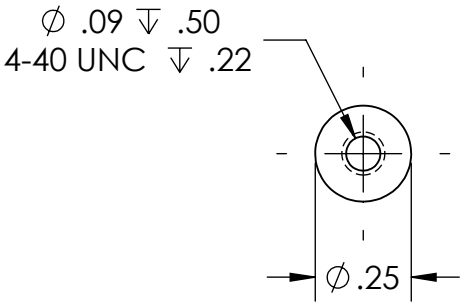
UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 2:1 SHEET: 1 OF 1







DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum

GPS

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

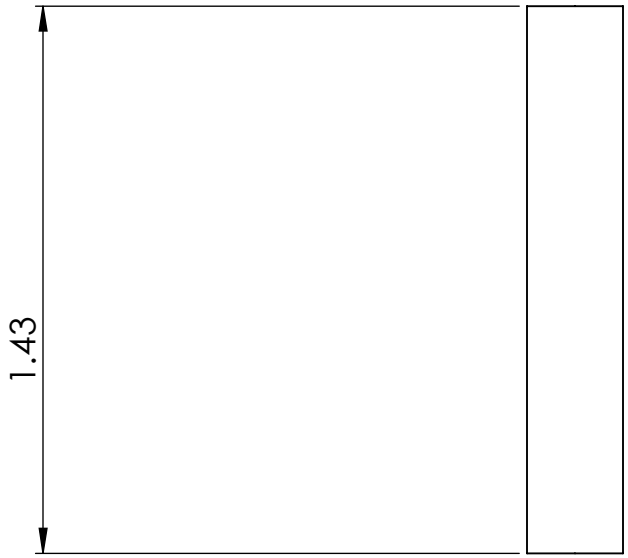
DRAWN BY: Brian Kisling DATE: 5/10/2015

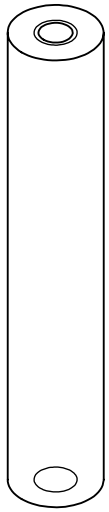
FILE NAME: 01-04 Support Rod-V3.SLDPRT

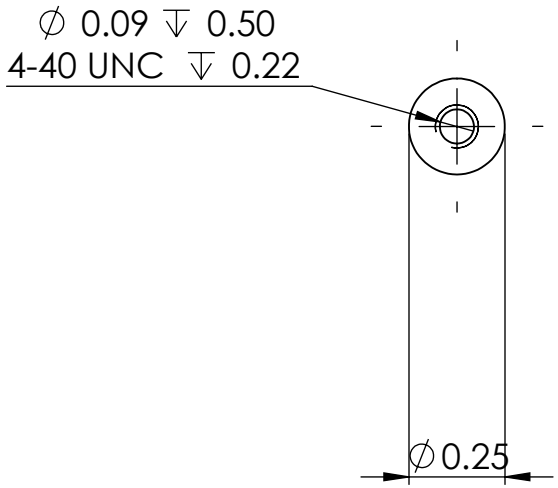
UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 2:1 SHEET: 1 OF 1







DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum

GPS

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

DRAWN BY: Brian Kisling DATE: 5/10/2015

FILE NAME: 01-04 Support Rod-V2.SLDPRT

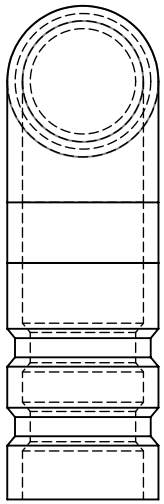
UNIVERSITY OF IDAHO
ME DEPARTMENT

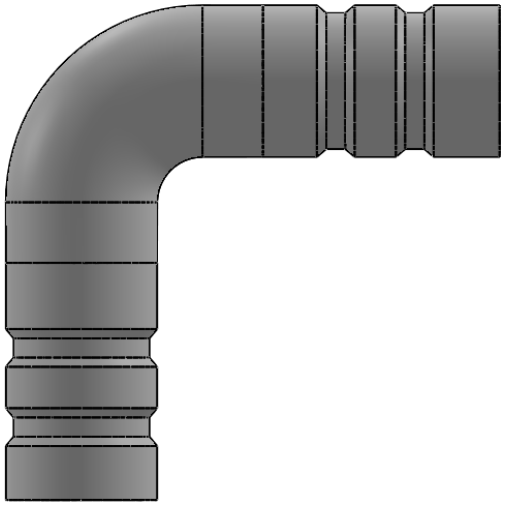
PART #:

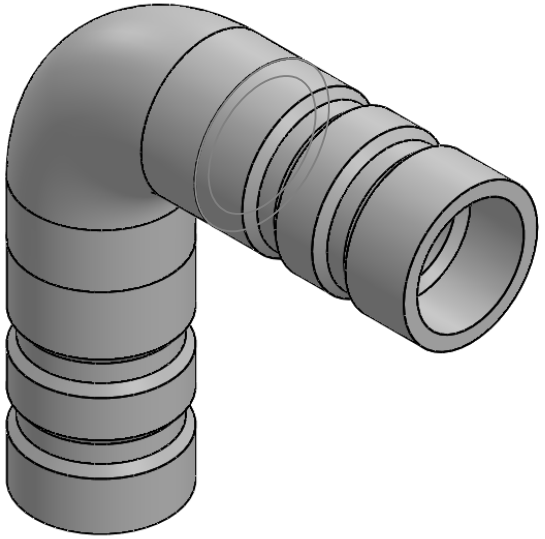
QTY:

SCALE: 2:1

SHEET: 1 OF 1







DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL:

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

DRAWN BY: DATE: 3/3/2015

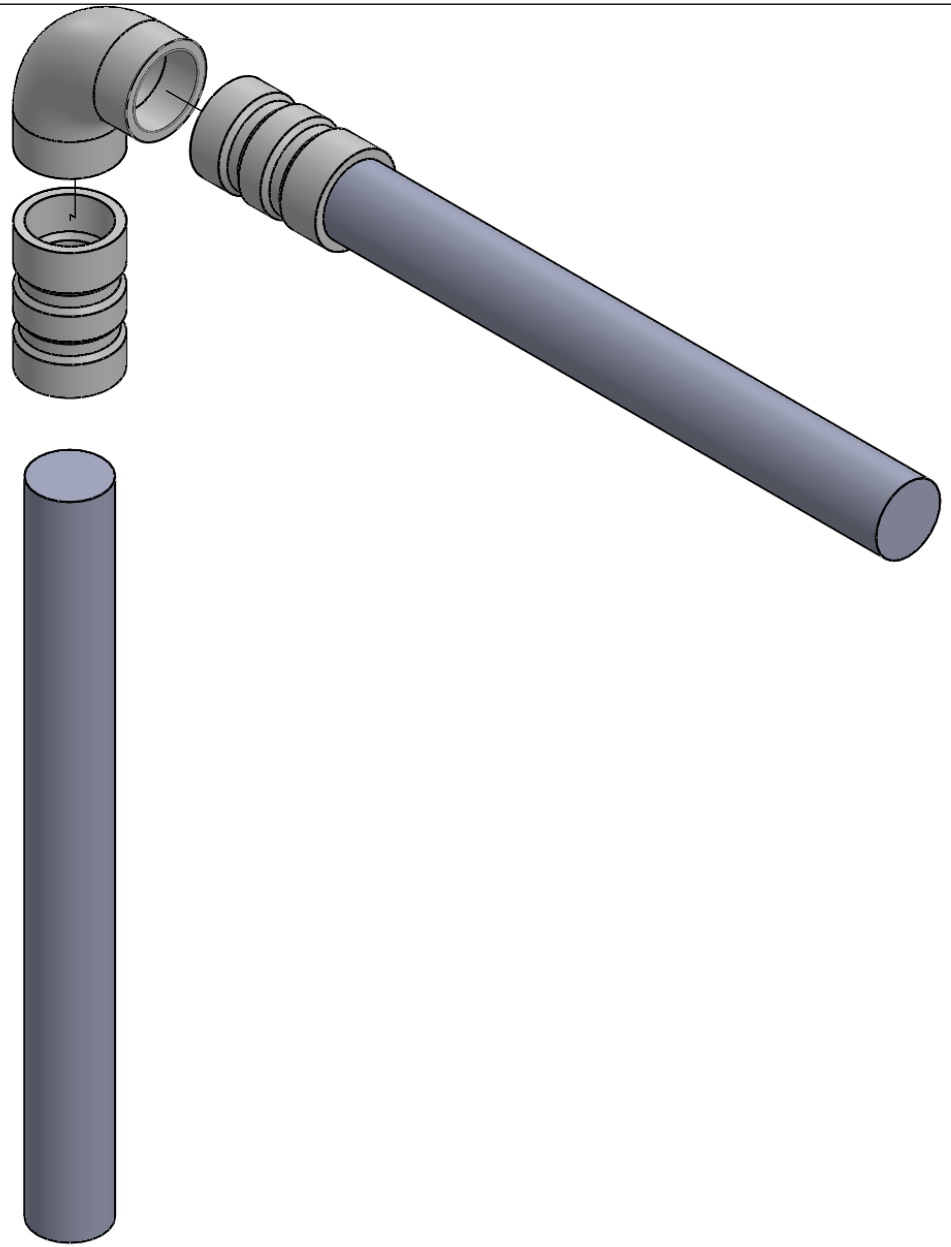
FILE NAME: Assem1.SLDPRT

PROJECT NAME

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 10:1 SHEET: 1 OF 2



DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL:

PROJECT NAME

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

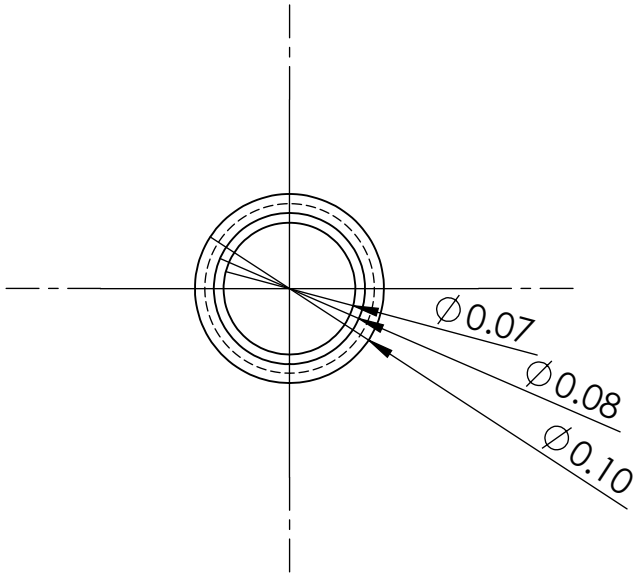
DRAWN BY: DATE: 3/3/2015

FILE NAME: exploded.SLDPRT

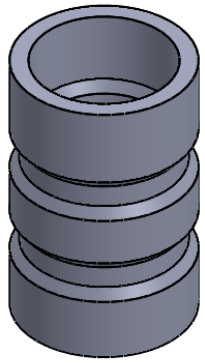
UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

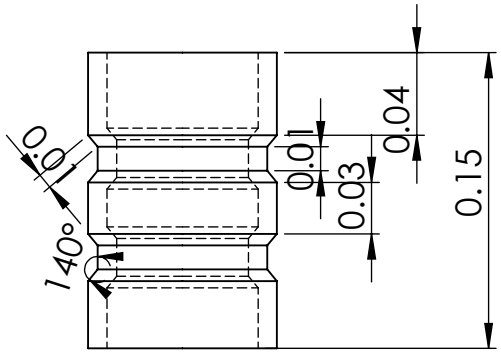
SCALE: 10:1 SHEET: 2 OF 2



TOP



ISO



FRONT

DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL:

DEFAULT TOLERANCES:

LINEAR:
X. ± .25
X.X ± .1
X.XX ± .01
X.XXX ± .002

ANGULAR:
X. ± 2
X.X ± 1
X.XX ± 0 30'

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

DRAWN BY: Effat Takaleh DATE: 3/3/2015

FILE NAME: 2.SLDPRT

90 elbow connector

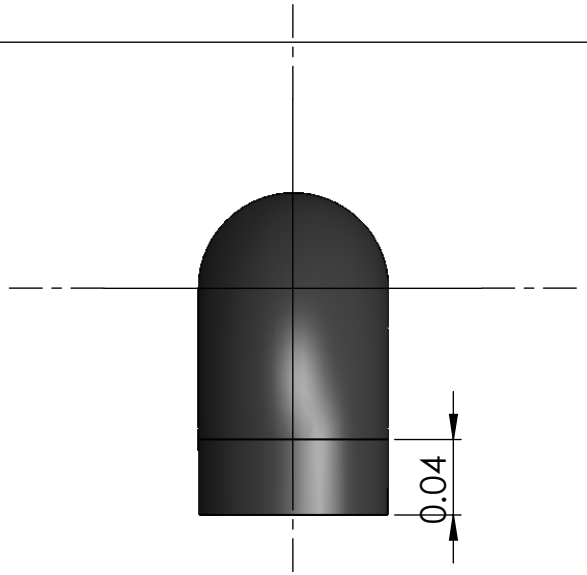
UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #:

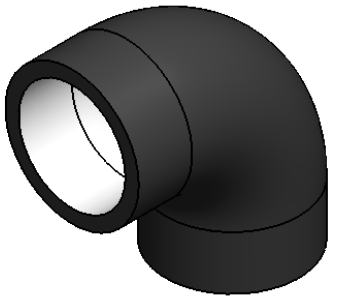
QTY: 2

SCALE: 10:1

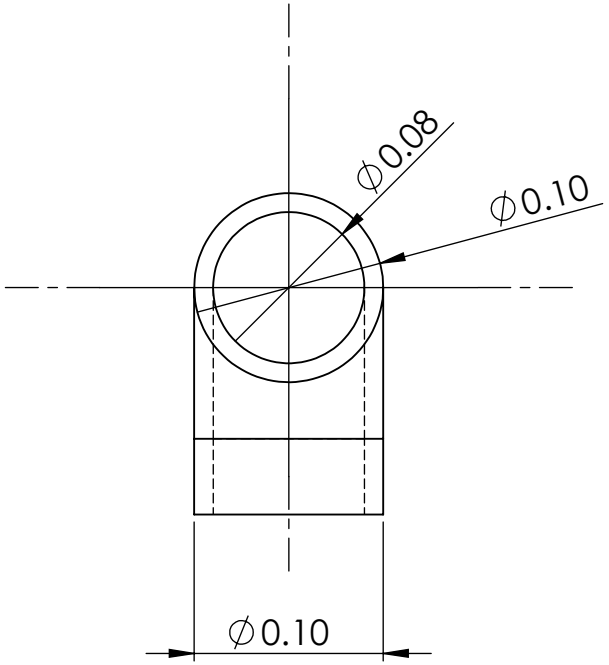
SHEET: 1 OF 1



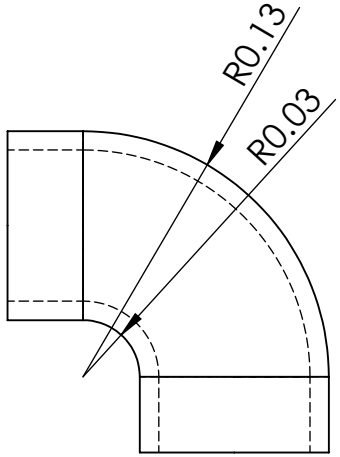
TOP



ISO



FRONT



RIGHT

DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL:

90 Elbow angle

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION:

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

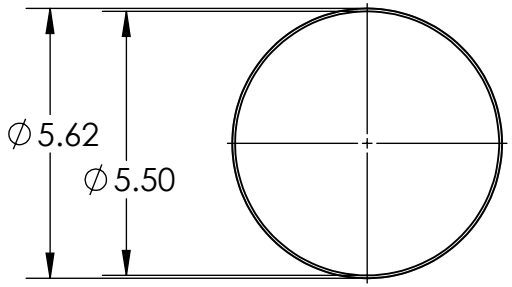
DRAWN BY: DATE: 3/3/2015

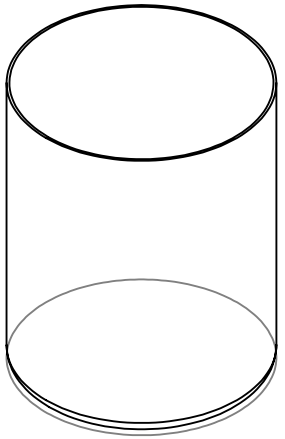
FILE NAME: 90 degree elbow.SLDPRT

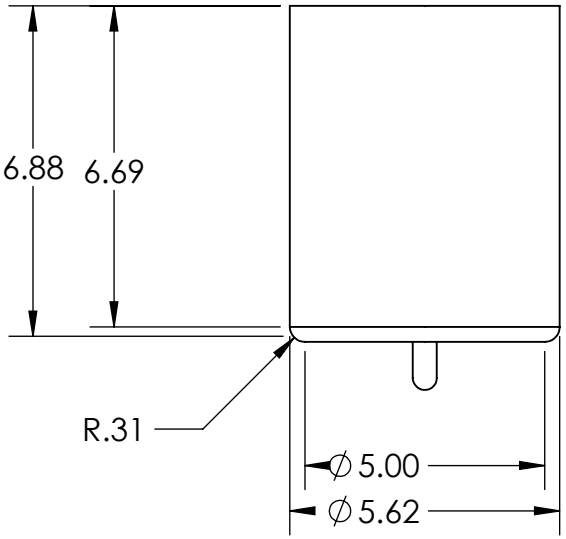
UNIVERSITY OF IDAHO
ME DEPARTMENT

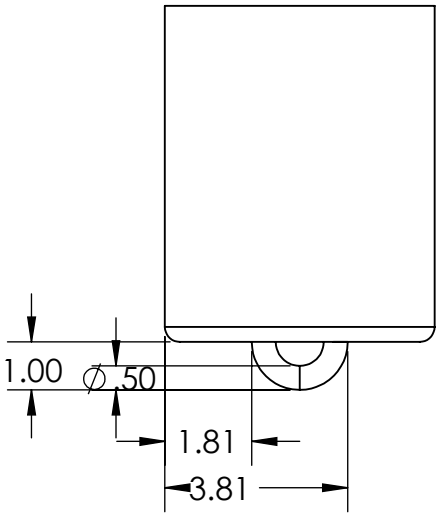
PART #: QTY:

SCALE: 10:1 SHEET: 1 OF 1









DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL:

GPS

DEFAULT TOLERANCES:

LINEAR:
X. ± .25
X.X ± .1
X.XX ± .01
X.XXX ± .002

ANGULAR:
X. ± 2
X.X ± 1
X.XX ± 0 30'

DESCRIPTION: PTCPU canister

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

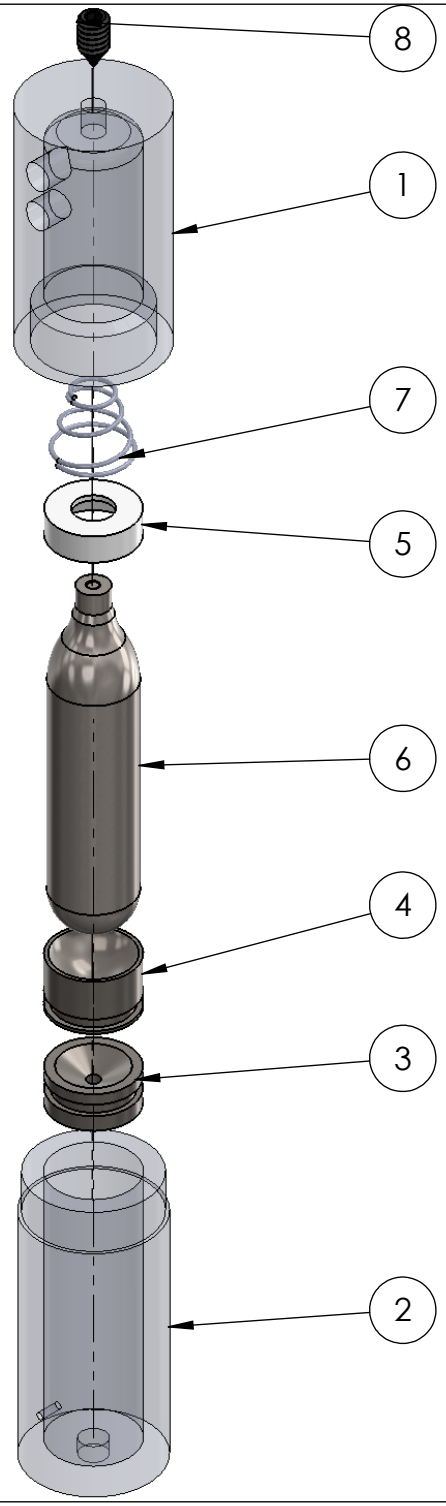
DRAWN BY: Brian Kisling DATE: 12/11/2014

FILE NAME: canister_V1print.SLDPRT

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: QTY:

SCALE: 1:4 SHEET: 1 OF 1



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	1-01	Main Casing with Side Exhaust	1
2	1-02	Bottom Section	1
3	1-03	Powder Holder	1
4	1-04	Cylinder Slug	1
5	1-05	Spring Holder	1
6	1-06	CO2 8g Gas Cylinder	1
7	1-07	Spring - 16925K8	1
8	1-08	Puncture Screw - 92785A437	1

DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Aluminum, SS, Plastic

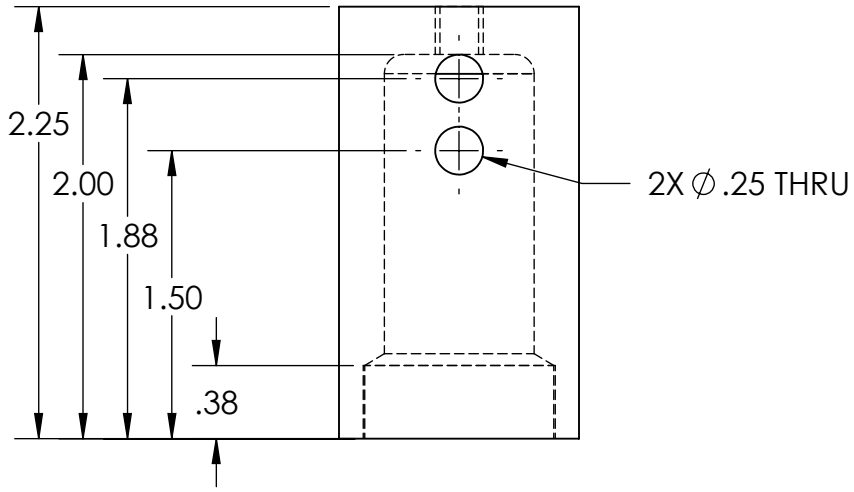
DEFAULT TOLERANCES:
LINEAR: X. ± .25, X.X ± .1, X.XX ± .01, X.XXX ± .002
ANGULAR: X. ± 2, X.X ± 1, X.XX ± 0 30'

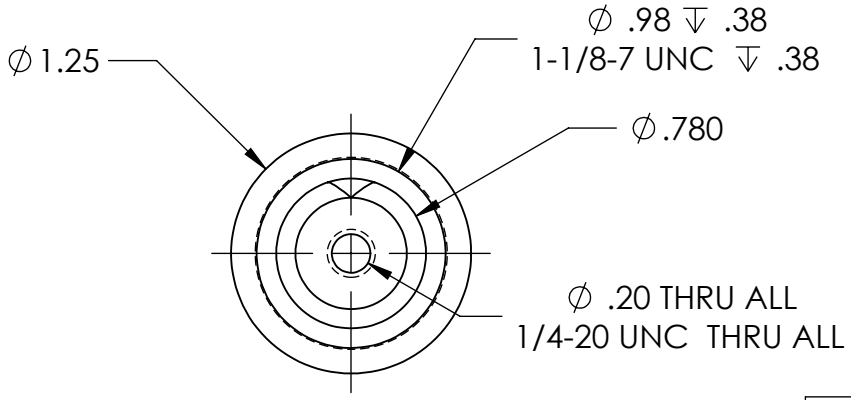
DESCRIPTION: Deployment System Assembly
CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX
DRAWN BY: Forrest Austin Tanner DATE: 5/11/2015
FILE NAME: 12g Deployment with Side holes.SLDPRT

Deployment System

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: 1-00 QTY: 1
SCALE: 1:1 SHEET: 1 OF 6





DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: 6061-T6 (SS)

Deployment System

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION: Main Casing with Side Exhaust

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

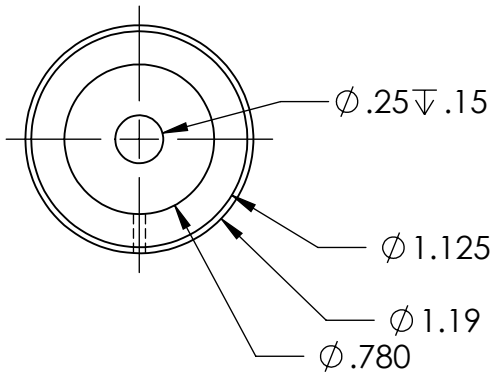
DRAWN BY: Forrest Austin Tanner DATE: 5/11/2015

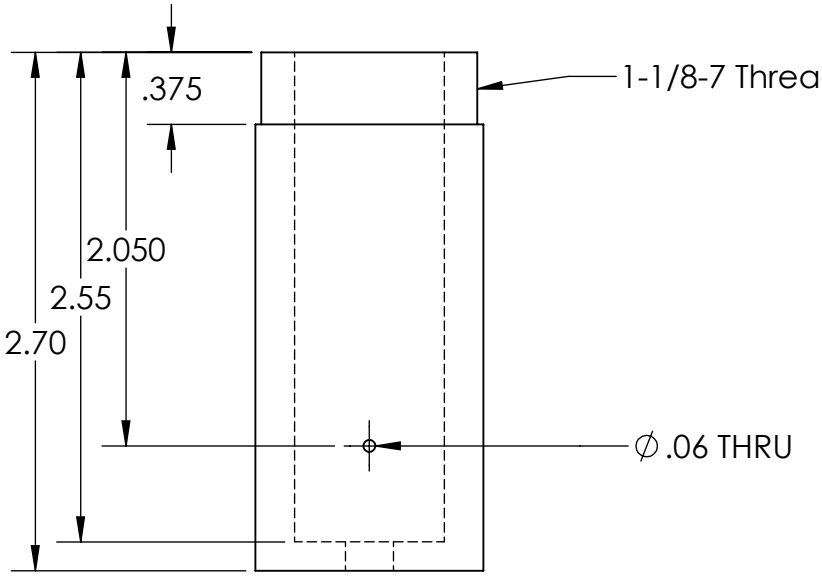
FILE NAME: 12g Main Casing with Side exhaust.SLDPRT

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: 1-01 QTY: 1

SCALE: 1:1 SHEET: 2 OF 6





DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: 6061-T6 (SS)

Deployment System

DEFAULT TOLERANCES:

LINEAR: X. ± .25, X.X ± .1, X.XX ± .01, X.XXX ± .002

ANGULAR: X. ± 2, X.X ± 1, X.XX ± 0 30'

DESCRIPTION: 12g End Cap

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

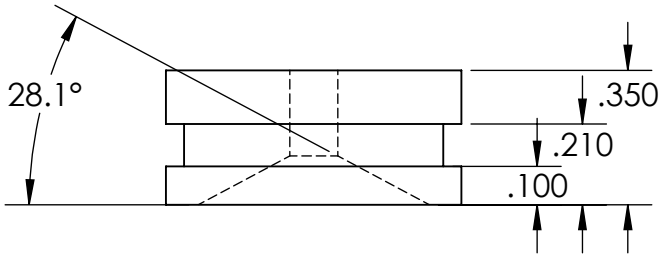
DRAWN BY: Forrest Austin Tanner DATE: 5/11/2015

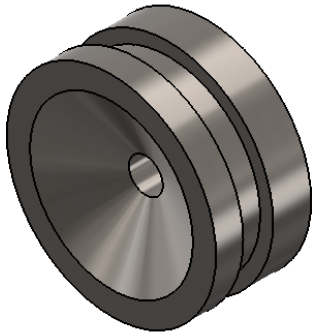
FILE NAME: 12g End Cap.SLDPRT

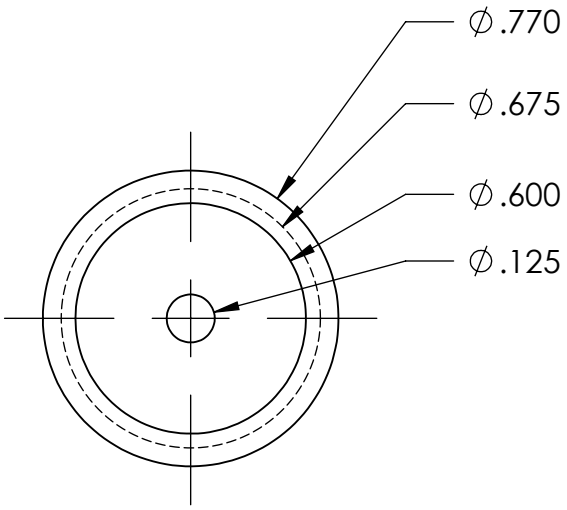
UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: 1-02 QTY: 1

SCALE: 1:1 SHEET: 3 OF 6







DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: AISI 316 Annealed Stainless Steel Bar (SS)

Deployment System

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION: Powder Holder

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

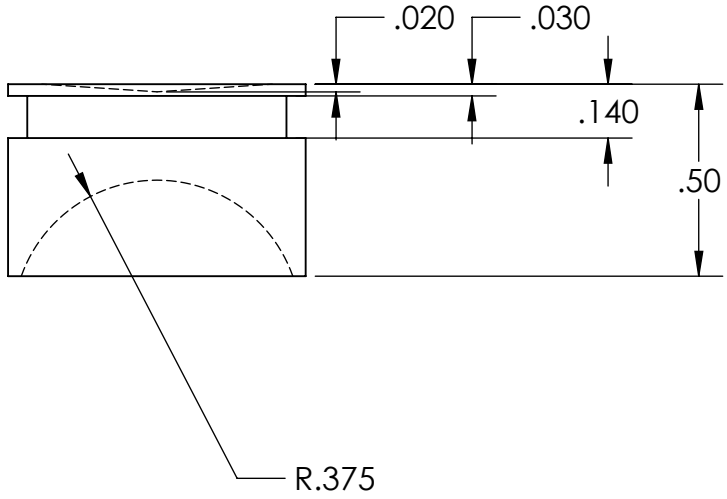
DRAWN BY: Forrest Austin Tanner DATE: 4/29/2015

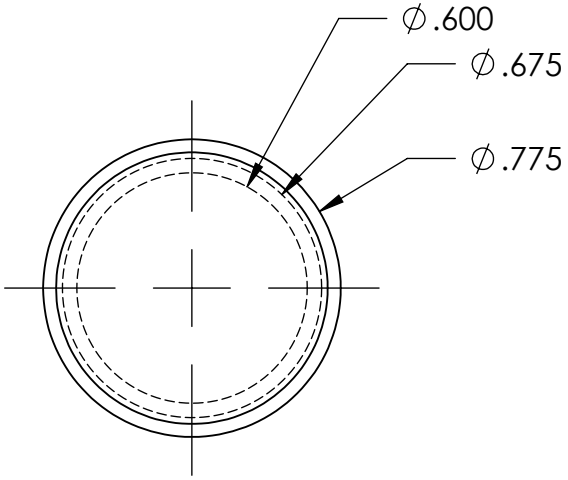
FILE NAME: Pyro Holder.SLDPRT

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: 1-03 QTY: 1

SCALE: 2:1 SHEET: 4 OF 6





PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UNIVERSITY OF IDAHO, ME DEPARTMENT. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UNIVERSITY OF IDAHO, ME DEPARTMENT IS PROHIBITED.

DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: AISI 316 Annealed Stainless Steel Bar (SS)

Deployment System

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION: Cylinder Slug

CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX

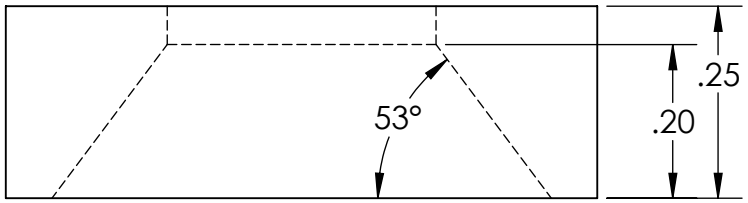
DRAWN BY: Forrest Austin Tanner DATE: 4/29/2015

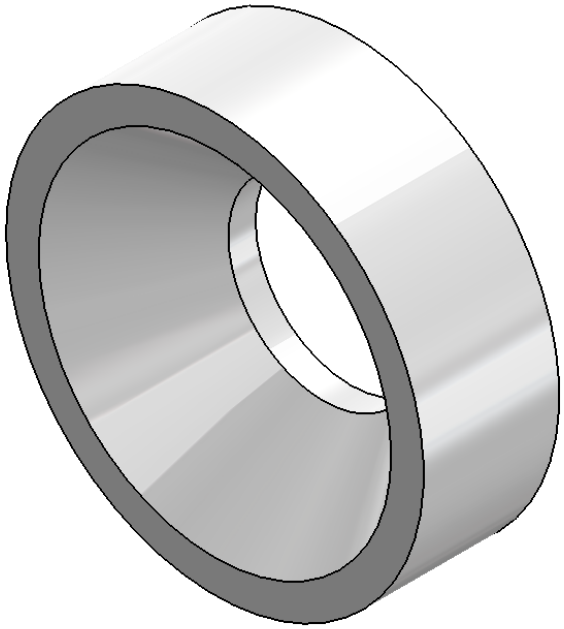
FILE NAME: Cylinder Slug.SLDPRT

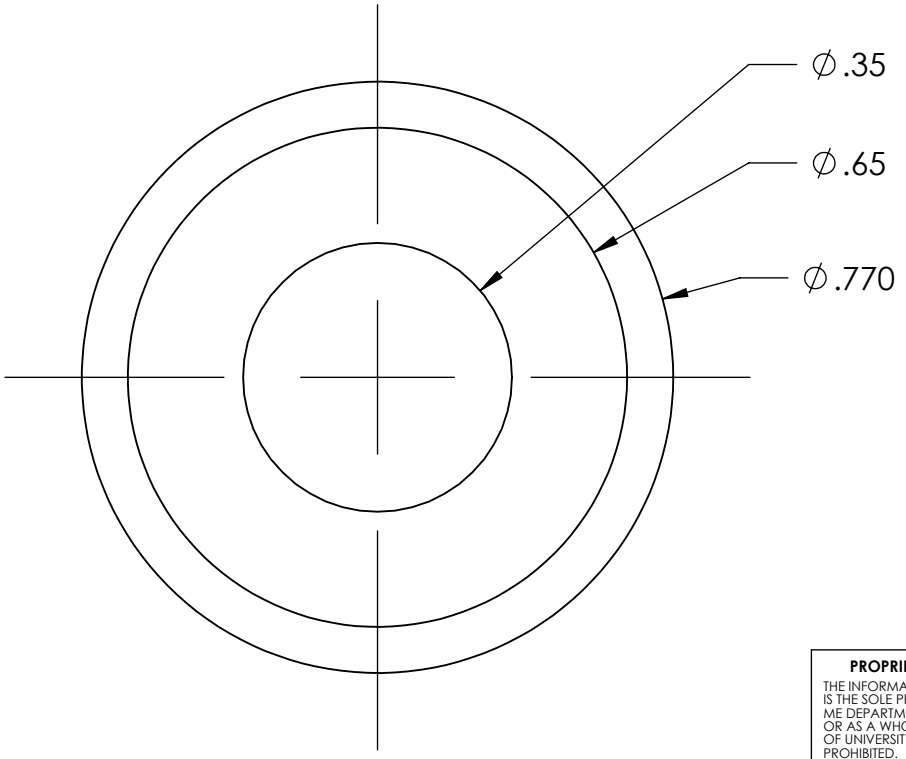
UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: 1-04 QTY: 1

SCALE: 1:1 SHEET: 5 OF 6







DIMENSIONS ARE IN INCHES
THIRD ANGLE PROJECTION

MATERIAL: Plastic or Aluminum

DEFAULT TOLERANCES:

LINEAR:	ANGULAR:
X. ± .25	X. ± 2
X.X ± .1	X.X ± 1
X.XX ± .01	X.XX ± 0 30'
X.XXX ± .002	

DESCRIPTION: Spring Holder
CHECKED BY: XXXXXXXXXX DATE: XX/XX/XX
DRAWN BY: Forrest Austin Tanner DATE: 4/29/2015
FILE NAME: Spring holder.SLDPRT

Deployment System

UNIVERSITY OF IDAHO
ME DEPARTMENT

PART #: 1-05 QTY: 1
SCALE: 2:1 SHEET: 6 OF 6

Appendix D. Budget Summary

Tentative Purchases for Budget					
Name	**Quantity**	**Unit Cost**	**Total Cost**	**Vendor**	**Comments**
EE Budget Estimate					
Irdium 9603	1	400	400		
Iridium antenna	1	200	200		**SPS-ANT-10058**
Iridium data	N/A	100	100		
Wireless Pressure Sensors	4	25	100		
Wireless Temperature Sensors	4	25	100		
Arduino Mega Pro (customized REV1)	1	150	150		
Arduino Mega Pro (customized REV2)	1	150	150		
GPS unit	1	200	200		
GPS antenna	1	100	100		
Batteries	2	150	150		
		EE Total	**1650**		
ME Budget Estimate					
Parafoil	2	25	50	HobbyKing	http://www.hobbyking.com/hobbyking/store/__14284__HobbyKing_Paraglider_Parafoil_2_15m.html
Servos	3	35	105		
Aluminum plate	N/A	10	150	Alcobra	http://alcobrametals.com/product/AS.188E
Aluminum bar	N/A	50	50	Alcobra	
Miscellaneous mechanical supplies	N/A	150	150		
Split Rings			5		
Fishing Line			2		
Canister			10		
		ME Total	**522**		
Testing					
Balloons	2	100	200		
Helium	2	125	250		
Travel	1 suburban/van	300	300		$60/day plus gas for approx. 250 mi
		Testing Total	**750**		
Travel Expenses					
Plane Tickets	8	187	1496		
Per diem first day	8	60	480		
Per diem second day	8	60	480		
Per diem third day	8	60	480		
Mileage	2	99.9	199.8		
Rental Car (per day)	3	100	300		
		Travel Total	**3435.8**		
	Estimate		**6357.8**		
	Total Requested		**8000**		